TECHNISCHE
UNIVERSITÄT
DARMSTADT

VIEWER-CENTRIC MOBILE SERVICES

A Framework and a Query Model

Vom Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation

von

Farid Zaid, M.Sc.

geboren am 2. Juni 1977 in Selet Alharthia, Palästina

Erstreferent: Prof. Dr.-Ing. Ralf Steinmetz
Koreferentin: Prof. Klara Nahrstedt

Tag der Einreichung: 15.09.2010
Tag der Disputation: 16.12.2010

Darmstadt, 2011
Hochschulkennziffer D17

إلى أمّي، أبي، إخوتي و زوجتي أهدي هذا العمل.

ABSTRACT

With the increased integration of high-end GPS units in mobile devices, combined with the integration of orientation sensors like digital compasses and accelerometers, a new class of mobile Location-based Services (LBS) is emerging: the *viewer-centric mobile services*. With focus on *viewer-centricity*, viewer-centric mobile services aim at providing the user with information that are not only relevant to her current location, but also relevant to her *field-of-view*. Therefore, a viewer-centric can tell the user where a restaurant is located in her viewing range, what food menu it is offering etc., all by pointing the mobile device to scan the surrounding environment.

The main goal of this thesis is to *develop a framework for viewer-centric mobile services and a query model*, with main focus on modeling viewer-centric queries from the GPS and orientation sensors existing in modern mobile devices. As these sensors, regardless of their fabrication quality, do suffer from runtime uncertainties, e.g. blockage of GPS signal by buildings, distortion of magnetic field due to nearby metallic structures, a key concept of the proposed framework, *iVu.KOM*, is to recognize the effects of such uncertainties on the overall accuracy of the modeled queries and to offer a means to correct the queries, when needed.

As the view of the user is also affected by obstacles, e.g. buildings, that exist in the user's surrounding and limit the viewing range, the framework aims as well at efficient execution of the viewer-centric queries against the world geometry model. For this purpose, the proposed *iVu query model* suggests to include in the query result both the retrieved points of interest and a simplified description of the surrounding geometry. This query model allows executing the queries promptly on the client and with minimal need to contact the server, even when the visibility changes due to user's motion.

To evaluate the feasibility of the proposed framework and model, a prototypical implementation of iVu.KOM was realized for a modern mobile platform featuring the required sensors and toolkit support. The prototype was evaluated under real world conditions in carefully selected test environments.

Additionally, the thesis investigates the usage of *participatory sensing* for positioning purposes. Here, WBroximity is proposed as a positioning solution that uses both WLAN and Bluetooth fingerprints, which are collected through participatory sensing. This aims at cutting the extra costs needed to employ special personnel for this task, and allowing the system coverage to

expand to wherever participants reach. WBroximity was evaluated through simulation and a prototypical realization. Moreover, WBroximity can be seamlessly used by iVu.KOM as a location provider, specially in indoor environments where other location providers, like GPS, can fail.

ZUSAMMENFASSUNG

Betrachter-zentrierte Mobile Dienste (Viewer-centric Mobile Services) bezeichnen eine neue Generation von Ortsbezognen Diensten (Location-based Services - LBS), wobei die Informationen bzw. Dienste anhand nicht nur des Ortes des Benutzers, sondern auch seiner Blickrichtung angepasst sind. Diese Entwicklung ist heutzutage dank der zunehmenden Integration von GPS Empfängern und Orientierungssensoren wie Kompassen und Beschleunigungssensoren in den modernen mobilen Geräten möglich. Durch die Kombination von diesen Sensoren lässt sich die Blickrichtung einfach modellieren. Damit wird es möglich, solche Anfragen wie "was bietet heute dieses Restaurant´´ einfach durch das Zeigen auf das Gebäude durchzuführen.

Trotz der Fortschritte in der Anfertigung von in den mobilen Geräten eingebetteten Sensoren, leiden diese Sensoren unter unvermeidbaren dynamischen Fehlerquellen. Zum Beispiel wird der Empfang eines GPS-Signals normalerweise durch die Gebäude verhindert. Ebenso unterliegt ein Kompass Störeinflüssen durch das nebenstehende Metall und die elektrischen Leitungen. Somit wird die Genauigkeit einer Betrachter-zentrierten Anfrage, die auf Basis dieser Sensoren aufgebaut ist, nachlassen.

In dieser Arbeit wird daher ein Framework sowie ein Anfragemodell zur Unterstützung von Betrachter-zentrierten Mobilen Diensten entwickelt. Als Hauptziel hat dieses Framework, den Einfluss von Sensorenungenauigkeiten auf die modellierten Anfragen zu erkennen und dagegen zu reagieren. Da die Sichtbarkeit von Objekten auch allgemein von der umliegenden Geometrie beeinflusst ist, visiert dieses Framework an, die Anfragen gegen ein Geometriemodell effizient auszuführen. Dafür wird das *iVu Model* als ein Anfragemodell entwickelt, das es ermöglicht, in der Antwort der Anfrage sowohl den Inhalt (Points of Interest- POI's) als auch eine vereinfachte Darstellung der Geometrie einzubinden. Da die Geometrie auch schon im Model enthalten ist, muss das mobile Gerät signifikant seltener den Dienst kontaktieren, um die Anfrage auszuführen.

Die Realisierbarkeit des vorgeschlagenen Frameworks, *iVu.KOM*, wird durch eine prototypische Realisierung auf Basis der Android-Plattform gezeigt. Der Prototyp wurde mit echten Sensordaten und unter realen Bedingungen getestet.

"He who does not thank people,
does not thank God."

— Prophet Muhammad P.B.U.H

ACKNOWLEDGMENTS

While I added the acknowledgments right at the end of my thesis writeup, I have been all the time thinking about this.

I am very glad to thank Prof. Ralf Steinmetz for believing in me and giving me the chance to do my PhD at the Multimedia Communications Lab (KOM). Thank you so much for the patience, the great support, guidance and encouragement which all together led to the conclusion of this work!

I am also full with gratitude to the priceless feedback provided by Prof. Klara Nahrstedt and the time she took to review my work. Thank you very much!

A very sincere gratitude goes as well to Dr. Parag Mogre, being a dear colleague and a wonderful team leader. Parag, without you this would have been difficult to finalize! Your enthusiastic support, wide horizon and excellent guidance have all shaped my ideas over the last few years. Thank you very much!

I am also very thankful to Prof. Matthias Hollick, whose feedback helped specially through the initial phases of my PhD to narrow down my focus and identify the first steps towards the final goal. Thank you so much!

I thank as well my colleagues in our team, Johannes, Matthias, Andreas, and Diego. I am very fortune because I worked in such a great team. Your comments and cooperation have enriched my work and made it such a success. Thank you so much!

I am also very happy to thank all the other colleagues at KOM. You have all made my three and a half years at KOM full with warmness and cheerfulness. Thank you all very much!

I never forget as well the kind help and unconditional support provided by my supportive friend Prof. Abdulmotaleb El Saddik. Thank you as well for the nice moments and fruitful discussions!

Last but not least, I am very thankful for the support provided by the German Research Foundation (DFG) within the Research Training Group 1362 ´´Cooperative, adaptive and responsive monitoring in mixed mode environments´´. The support was generous not only financially but also on the side of professional and personal development.

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

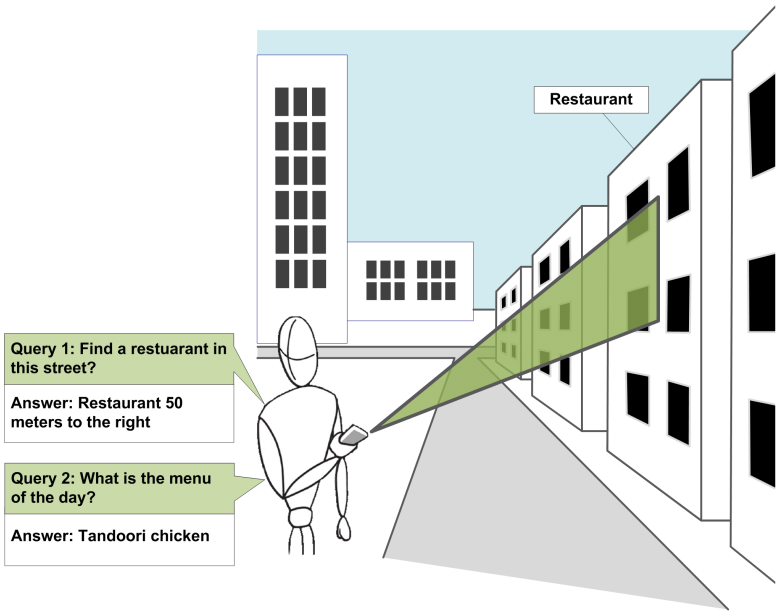


Figure 1: Scenario: a viewer-centric mobile guide

Ali is a researcher participating at a scientific event in an unfamiliar city. After the end of the event, Ali decides not to lose the chance to explore the city attractions. While he is hanging around, Ali starts feeling hunger, and he wishes to get to the nearest restaurant. To deal with such a situation, Ali thinks that his smart phone will be very useful as it provides him with access to a *viewer-centric* guide. As shown in Figure 1, firstly, Ali points his device to scan the street to find a restaurant in his viewing range (Query 1 in figure). Ali is lucky, as the service identifies a matching restaurant and tells him of the direction and distance to reach it. Before going into the restaurant, Ali wishes however to know the menu of the day. For this purpose, he points his device again towards the restaurant (Query 2 in figure), and gets the menu displayed on his device. Ali is satisfied, as he found his destination in a very short time and without too much interaction with his device.

The aforementioned scenario will be possible due to the advancements in communication and computing technology which

are accompanied with increasing enrichment of end-user mobile devices with different kinds of sensing capabilities. In contrast to the contemporary Location-based Services (LBS), which provide information or services customized to the user's current location, the given scenario sketches a kind of *viewer-centric mobile services*. Viewer-centric services go one step further beyond LBS by adjusting the service not only to the user's location, but also to her view. Such *viewer-centricity* is achieved by exploiting the orientation sensors, i.e. the digital compass and the 3-axis accelerometer, in addition to the location sensor (e.g. GPS) that are embedded in modern mobile devices. A viewer-centric service has a three-fold advantage over classical LBS:

- the service is customized not only to the user's current location, but also to her *viewing direction*. For example, while searching for a nearby restaurant, instead of getting the answer "a restaurant is found within a 50 meters range" as provided by classical LBS, the answer by the sketched service will be "a restaurant is 50 meters away, to the right of viewing direction". Obviously, this gives the user a better aid to orient herself towards the destination.

- the service is customized as well to the user's current *viewing range*. In classical LBS, the search is normally restricted to some search radius. Consequently, some of the search results, although lying within the specified search radius, will lie outside the viewing range of the user because they may be blocked by existing obstacles (like buildings). In contrast to this, the sketched scenario suggests that a returned result does really lie within the user's visible range. This in turn increases the relevance of the results (e.g. restaurants that the user can see in the same street will be more relevant than those that are hidden by buildings, even if the hidden ones have closer distance). Consequently, the user becomes more physically immersed in the geographical region being searched.

- the service offers a viewer-centric input modality which is more appropriate for mobile settings. Typically, existing LBS (e.g. Google Mobile [11]) offers an interface to enter the search query in an input box. The user has then to pick out a relevant search result, probably to iterate the search, and eventually to navigate to the desired destination. In contrast to this, the scenario suggests a *pointing* metaphor to query the service. The user simply points towards the area she wishes to search in, or towards the objects she wishes more information about. The returned results have then implicit spatial association with the area or objects being pointed

at. Obviously, this interaction is more relevant for a mobile setting which needs to have immediacy, quick interface navigation and precise use of limited mobile device screen space.

Without doubt, with viewer-centric services a user will have an intuitive access to information about objects and services in the user's *locality*. Thinking of the enormous growth of the mobile LBS (in 2010, LBS revenues in Europe will reach €622 million, accounting for 1.8 % of the non-voice services [29]), and keeping in mind that a big portion of the mobile LBS usage has local intent (mobile search with local intent is expected to reach 35% of the total mobile search [7]), viewer-centric services can play a sigificant role in the consumer industry sectors and provide good opportunities for lucrative local businesses.

However, a still missing link to reach the actual potential of viewer-centric services is the modeling of viewer-centric queries where a level of query accuracy and query efficiency is provided. While query accuracy is mainly affected by the uncertainties of the sensors (e.g. GPS, compass and accelerometer) that are used to build the viewer-centric queries, the query efficiency is mainly affected by the needed communication with the service and executing the query against a model that represents the physical world being searched.

This dissertation targets therefore at a framework and a model for viewer-centric queries with focus on supporting both query efficiency and query accuracy. Based on an open architecture which allows plugging different query models and different sensors, the framework is designed to offer viewer-centric services both in indoor and outdoor environments.

1.2 HISTORICAL BACKGROUND

In 1995 network operators in the US, Europe and Japan, were forced by legislations to provide the location of the user when she calls the emergency service number (e.g. 911 in US and 112 in Europe) [62]. Since then, LBS was widely adopted by network providers as part of their service portfolio. Nowadays, besides public safety, LBS features many useful consumer services like navigation, location-based search and location-based advertising.

Inspired by the advances in GPS and sensor technology, Egenhofer [26] predicted in 1999 a set of geographical applications that combine both location and orientation information. One example included the *GeoWand*; a device that allows users to get information about physical objects by just pointing towards them.

In 2003, Faisal [30] studied the use of *pointing* as a query modality for LBS. His work focused on analyzing and correcting the er-

rors arising due to the discrepancy between the user's actual Line-of-Sight (LoS) and the pointing vector that is (virtually) formed by the pointing device. However an evaluation of the effect of GPS errors on the overall pointing quality was not conducted.

Gardiner et. al. [33] discussed in 2003 the implementation of a mobile viewer-based directional query system. They utilize the location and orientation information to construct a directional query that mimics the user's field of view. The query is executed against a world geometry model stored in a spatial database, thereby filtering out all POI lying outside the viewing port and hidden by geometry. However, no evaluation of GPS and orientation errors was provided.

Simon [57] proposed in 2008 a framework for spatially-aware mobile applications. Similar to [33], a query is executed against a geometry model to identify the visible points of interest. The query result is also extended with a billboard-based description of the geometry surrounding the user. The query result (called by the author as the *Local Visibility Model*) is then sent to the client, which uses it locally to run orientation-aware queries. Including the geometry description in the query result has the advantage that queries can be executed promptly on the client, therefore offering the user a better real-time experience. Although the author evaluated the effect of GPS and compass errors on the query accuracy, the author acknowledges that the used geometry simplification can lead to additional errors due to the perspective mismatch between the rectangular billboards and the actual buildings' facades. Additionally, it is noticed that the billboards are location-dependent; once the user moves, the billboards have to be re-computed on the server, which may have efficiency downsides.

More recently, commercial applications like [3, 5] have provided platforms for mobile augmented reality, where digital information in the viewing direction are overlaid on top of the mobile camera. Such applications however still lack automatic means to limit the search radius. Therefore, the users have to manually adjust their viewing range. Additionally, common to these application is that they only support content retrieval. Creating new content (or POI) through pointing is generally not supported by these applications. Other applications like Google Mobile [11] (featuring a voice-based query input) and Google Goggles [2] (featuring a visual query input) still have long trip to go as they heavily count on the advancements in voice and image recognition technologies.

1.3 REQUIREMENTS

Based on the aforementioned historical review of major works in the area of mobile viewer-centric services, the following functional and non-functional requirements can be identified:

1.3.1 *Functional Requirements*

The viewer-centric service should ideally support the following functionalities:

Point-to-find *Retrieve a POI along the LoS.*

For a given target object (e.g. a building) along the LoS, a user is able to retrieve the POI registered to that target. A POI can be any type of multimedia content (photos, videos, websites, RSS feeds etc.).

Point-to-tag *Create a POI along the LoS.*

For a given target object along the LoS, a user is able to create a POI and register it to that target. This functionality enables the in situ contribution of user-generated content (e.g. give reviews on-site to a sightseeing or blogging an event).

Point-to-search *Retrieve points of interest (POIs) within the Field-of-View (FoV).*

For a given area, a user is able to retrieve POIs that are lying within her FoV either in browsing-mode (i.e. retrieves all existing POIs) or in searching-mode (i.e. retrieve only POIs matching a given search criterion).

1.3.2 *Non-functional Requirements*

The following non-functional requirements are stressed here:

Query Accuracy *Meaning that a retrieved or created POI is really registered to the objected being pointed at.*

Sensors used to construct a viewer-centric query suffer in general from dynamic errors , e.g. location error due to buildings blocking the GPS signal, and compass error due to magnetic interference.

Query Promptness *Meaning that query results are presented timely to the user.*

Due to user's mobility, query results may lose their validity if there is a long *latency* in query processing or in communication. Query promptness can be even safety critical for some applications like navigation.

Query Efficiency *Meaning that running a query makes efficient use of the system resources.*

Efficient queries involve a high *performance* to *cost* ratio. While performance can be observed through the experienced query accuracy and query promptness, costs are associated with system resources needed for running the queries, typically including:

- *Link bandwidth* to communicate queries and their answers between the mobile clients and the service. Bandwidth constraints become a major design issue especially for *continuous queries* [68] for example when 'the user wants to know the closest restaurant as she moves along'. Clearly, the answer to such query needs to be updated as the user is changing her position, resulting in increased network traffic.
- *Computational power* both on the client and the server sides. At the client, queries need first to be built by collecting sensory data (e.g. from GPS, compass and accelerometer). Query answers are to be processed as well at the client so that they are presented to the end user's application. On the server side, queries are to be run against a spatial database which maintains the world geometry model.

Mitigating communication and computation costs implicitly leads to more *energy-efficient* queries which is highly appreciated in battery-powered mobile devices. Additionally, adopting strategies for efficient query processing enables the service to *scale* to a high number of users and queries per user while preserving acceptable accuracy and promptness per query.

1.4 THESIS GOALS

Having motivated the importance of mobile viewer-centric services and highlighted major works in this area, this thesis aims mainly at *developing a framework and a query model for mobile viewer-centric services*. In particular, the following steps are needed to achieve this goal:

- *state of the art...*

  To establish a baseline for the work in this thesis, the existing approaches for enabling viewer-centric mobile services are to be studied in detail.

- *developing a framework and a query model...*

  Based on the findings of analyzing the existing works, a framework for enabling mobile viewer-centric services is to be developed. The framework should be centered around a model for viewer-centric queries, i.e. queries that are constructed from location and orientation sensors and mimic the current view of the user. The model has two main design considerations: it allows for executing the queries efficiently, and it allows to recognize and react to the effects of location and orientation sensor uncertainties on the overall query accuracy.

- *evaluation...*

  The feasibility of the proposed framework and model is to be evaluated. A first step in the evaluation is to build a prototypical implementation of the framework using a suitable mobile platform. The second step will be to evaluate the performance of the framework under real world conditions, including real sensory data and geometry models.

1.5 THESIS OUTLINE

The rest of this thesis is structured as follows:

Chapter 2 briefly highlights the research fields intersecting with the topic of this thesis and describes the needed theoretical background.

Chapter 3 presents iVu.KOM; the proposed framework for viewer-centric services. It also explains the design of the iVu model; the proposed query model. Chapter 4 introduces the concept of the *skyline* which is used to run the viewer-centric queries. Chapter 5 discuses the design of an augmented reality feedback interface to recognize and react to sensor uncertainty. Chapter 6 presents the design details of a positioning service that can be used as a location provider for iVu.KOM, specially in indoor environments.

Chapter 7 describes the technical details of implementing the iVu.KOM framework on top of the Android platform. Chapter 8 describes the evaluation methodologies and discusses the results.

Chapter 9 reviews the fulfillment of the thesis goals and highlights main directions for future work.

2 BACKGROUND

This chapter briefly describes the research areas that are intersecting with the topic of this thesis. It then describes the theoretical background needed by the proposed approach.

2.1 RELATED RESEARCH AREAS

Figure 2 depicts the research areas which are intersecting with work presented in this thesis, i.e. the viewer-centric mobile services. In the sequel is a brief description of each area.

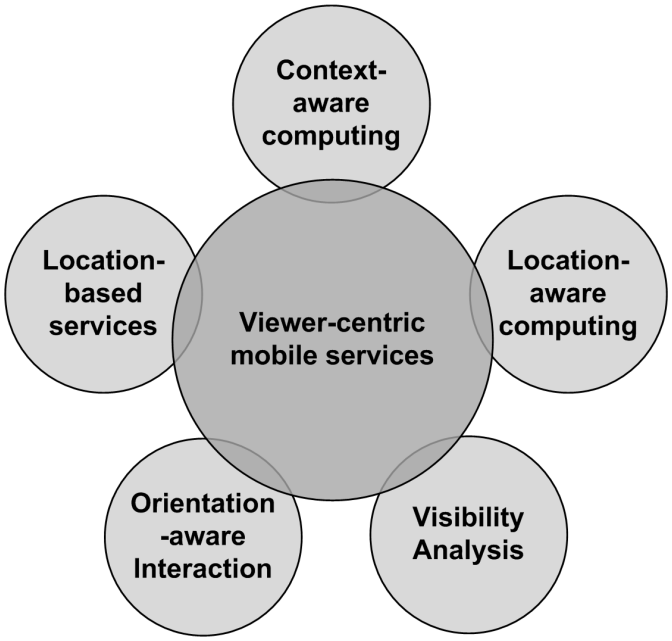


Figure 2: Related research areas

2.1.1 *Context-aware Computing*

Context has a broad semantic

Context can mean many things. Schmidt et al. [55] propose the following definition: "*a context describes a situation and the environment a device or user is in*". They distinguish two general categories: context related to human factors (e.g. habits, emotional state, social environment and current task), and context related to the physical environment (e.g. location, surrounding infrastructure, weather and motion of the user). This characterization is completed by a time axis, along which the context varies. Time is also an aspect of context (e.g. time of the day).

With this meaning of context, *context-aware computing* provides a computing paradigm which 'uses context to provide task-relevant information and/or services to the users, wherever they may be' [54]. For example, searching nearby leisure activities is easier when the application can detect the person's location. The results can be ranked by time of the day and the weather. On sunny days outdoor activities are pleasant, while going to the cinema is a better choice when it rains. Besides, knowledge of the user's preferences helps making the ranking more relevant.

In general, the behavior of a context-aware application can be one of following:

- adapting the information presented to the user and the way it is presented,
- automating and adapting the execution of services, and
- labeling the information with contextual metadata to facilitate information search and retrieval.

Viewer-centricity as a special type of context

From this characterization, it is obvious that the work presented by this thesis, i.e. the viewer-centric services, can be classified in the area of context-aware computing. Here, the user's current location and viewing direction form the contextual dimensions needed by the application to provide its context-aware behavior, i.e. presenting the information in the visible proximity.

2.1.2 *Location-aware Computing*

Location is probably the most discussed context dimension in literature. One may understand location as position, which is defined by coordinates (e.g. latitude and longitude), yet this is only one of its aspects. Position is a quantitative identification of a location. It can also be expressed qualitatively (e.g. "in the car"). In addition, it can be distinguished between absolute and relative expressions of location. Which form is more appropriate depends on the use case. For instance "close to the bus station Luisenplatz in Darmstadt" is more relevant than the position in GPS coordinates if one wants to know what bus to take.

With location-awareness it is referred to "the general ability of an application to infer about the physical location of its user, and to adapt its information offering, user interface or behavior in response to this information" [57]. In general, the location information itself can be obtained using one of the following positioning techniques:

- Self-positioning: here, the mobile device uses information obtained from its own sensors to compute its own position.

The most common example is based on using satellite systems (currently the only publicly available one is GPS). For this to work, the mobile device must have at least four satellites in its line-of-sight. It listens to the signals transmitted by the satellites, and computes its distance to each of them. It can then determine its 3D position using a triangulation technique. Domestic GPS have an accuracy of less than 10m [67]. Drawbacks of GPS positioning are its failure in case less than four satellites are in line-of-sight, specially indoors, and its high power-consumption.

- Network positioning: here the network nodes use signals received from the mobile device to compute the position of the device. For example, Cell Identification or Cell of Origin approximates the position of the phone to that of the cell site (antenna) through which it is connected to the network. The accuracy of this method is generally very low. Depending on the cell size, it ranges from 300m in cities to 10km in rural areas. It can be improved by measuring the signal strength which leads to an accuracy of 100m to 500m [25].

- Hybrid positioning: this, like Assisted GPS, combines GPS with network techniques. This compensates for GPS failures, drops energy consumption and increases accuracy, where it can be as good as 5m.

2.1.3 *Orientation-aware Interaction*

Orientation: more context to location

Orientation-aware interaction goes one step further beyond location-aware computing. It augments location information with orientation of the mobile device. This enables different novel interaction metaphors with the physical environment surrounding the user, e.g. retrieving information about objects by simply pointing the mobile device towards them [57, 34, 3, 5]. The orientation information itself can be obtained through the following sensors embedded in the mobile device:

- Accelerometer: 3D accelerometer-based tilt sensors measure the acceleration exerted on a small mass by gravity, in order to determine the inclination (roll and pitch) of the device [58]. Accelerometers can be used as motion sensors as well. This has permitted to develop pedometer applications (calculation of the running distance). Yet, such techniques are only applicable in some specific cases, when the motion is regular. In the general case, dead-reckoning cannot be done using only an accelerometer, since it can hardly distinguish between a rotation and a shift of the phone. Rotation sen-

sors (e.g. using gyroscopes) are therefore needed as well. However, gyroscopes are not available yet on mobile phones and are predicted to be mass produced by 2012 [17].

- Digital compass: magnetometers can determine orientation even if the user is at stop. They work by measuring the direction of the Earth's magnetic field, like traditional compasses, and must be combined with accelerometers to compensate for the bias introduced by the inclination. Their accuracy is 1°-3° [58]. Yet the error can be much more significant in presence of a nearby electrical or magnetic field.

2.1.4 *Location-based Services (LBS)*

Location-based Services (LBS) refer to *''a set of applications that exploit the knowledge of the geographical position of a mobile device in order to provide services based on that information."* [53]. Historically, LBS found their first application in public safety, where in 1995 network operators in the US, Europe and Japan, were forced by legislations to provide the location of the user when she calls the emergency service number (e.g. 911 in US and 112 in Europe).

Location-based Services combines more than one technology

Nowadays, LBS can be considered as an intersection of three technologies [19]:

- mobile networks as access technology,
- Geographic Information System (GIS) and spatial databases as service enabling technology, and
- the Internet as a transmission and communication infrastructure.

LBS can be classified into two main categories: "pull" services, which are requested by the user, and "push" services, which are triggered automatically when certain conditions are met. Another way to classify them is to consider their domain of application as follows [51]:

- navigation/routing
- information services (e.g. ATM locations)
- social and collaborative services (e.g. friend finder)
- emergency, safety and medical/health services
- asset/person tracking and fleet management
- transaction and billing
- mobile office

- entertainment

Location-based applications do not differ only in their purpose, but also in the way the user interacts with the device. In 1999, Egenhofer predicted that the technology would evolve quickly enough to enable soon a range of "spatial information appliances" [26]. He described geo sketch pads (which allow the users to attach notes and media contents to a location), smart compasses (which show the direction of given targets), smart horizons (which display an extended panorama by integrating objects that are behind the horizon), geo-wands (which allow users to get information about remote objects by pointing at them) and smart glasses (which augment reality by superimposing a digital image into the visual field).

2.1.5 *Visibility Analysis*

Visibility is a well-established area

Visibility analysis is one of the important features needed by Geographic Information Systems (GIS). For example, the line-of-sight function finds the visibility from the viewing point to a target point considering the 3D environment [34, 18]. The path from the viewpoint to the target point is one of the rays emitted from the viewpoint. The emitted rays are truncated at their intersection with the obstacles (e.g. building blocks) while on route to the target. The set of interaction points forms a convex polygon or volume representing the visibility area of a specific viewpoint. In a 2D environment, where the elevation of the objects is not taken into account, the line-of-sight analysis is simpler where it is only needed to track the intersection points between the rays and obstacles in the horizontal plane.

Visibility computing is also very crucial for computer graphics. Nowadays, there are many different visibility algorithms for various visibility problems. The earliest visibility algorithms aimed at finding which lines or surfaces are visible in a synthesized image of a 3D scene [18]. Nowadays, two widely spread visibility algorithms can be identified:

- the z-buffer: this is applied for finding the visible surfaces. This algorithm and its variants are mainly used in the area of real-time rendering, like computer games, specially that it is easily implemented in graphics hardware.
- ray shooting: this is applied mainly for computing visibility along a single ray, e.g. for generating global illumination.

It is to be stressed here, that it is not a goal of this thesis to build new visibility algorithms. The main aim is to devise means to accurately model the line-of-sight from mobile device embedded

sensors, and to evaluate the applicability of these models under real world conditions. The modeled line-of-sight can be used as input to any of the existing line-of-sight analysis algorithms.

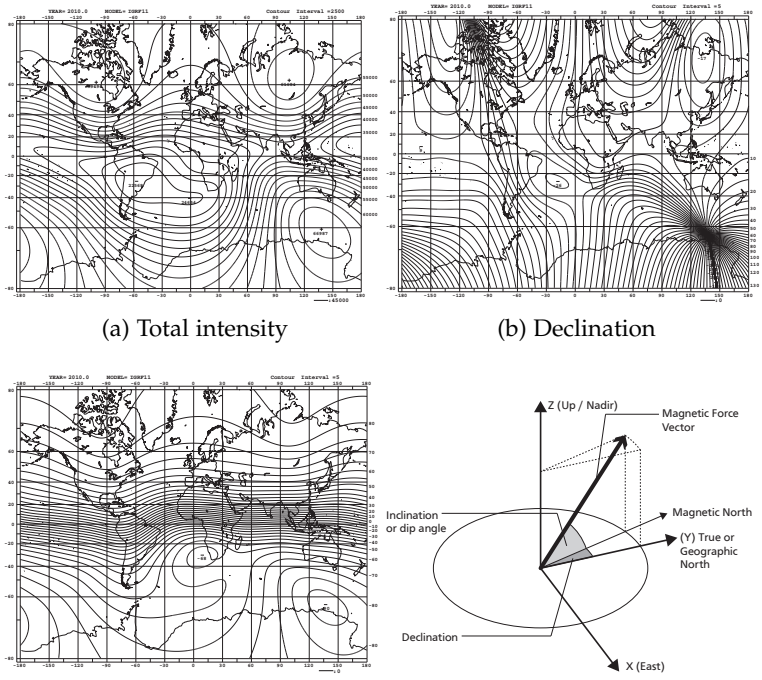
2.2 THEORETICAL BACKGROUND

This section describes the theoretical foundation of the work conducted in this thesis.

2.2.1 *The Earth's Magnetic Field*

Components of the Earths magnetic field

Several models do exist to measure the Earth's magnetic field. For example, the World Magnetic Model [48] defines the magnetic force vector at every location on earth. The *total intensity* defines the length of the vector. The vector is not aligned horizontally with earth's surface; there is a unique tilt for every place on earth. On the northern hemisphere the magnetic field points downwards, whereas on the southern hemisphere it points upwards. The angle that describes this tilt is called *inclination* and is also given by the model. The third important information the model provides is called *declination*. This is the angle one has to add to get a compass reading towards true or geographic North. Figure 3 visualizes these different components.



(a) Total intensity (b) Declination

(c) Inclination (d) Inclination and declination defined

Figure 3: Components of the Earth's magnetic field

2.2.2 *Digital Compass in Mobile Devices*

A compass is needed to tell the heading, i.e. direction in horizontal plane

The digital compass: mathematically, a compass measures the angle between it's own directional axis and the direction towards north. Depending on the context of use, there are several terms for that angle. One might find the terms *bearing*, *heading*, *yaw* and *azimuth*. All of them refer in a way to the angle a magnetic compass shows but there are slight differences in meaning due to different underlying coordinate systems that differ in reference system (local or world frame) and left or right hand orientation. Additionally, the use of north as reference can be confusing, due to the difference between the locations of the geographic and the geomagnetic north poles. The geographic poles of the Earth are fixed points that are given by earth's geometry. They do not change over time. On the other hand the magnetic poles are given by Earth's constantly changing geomagnetic field. In the remaining of this thesis, the term *heading* will be used to designate the compass direction from the geographic north.

The compass can be implemented in hardware using different techniques. For example, a Hall-Effect based compass measures the voltage that is induced in a conductor by the magnetic field. This is the kind of compass provided by the *Asahi Kasei Microsystems AK8976A* module found in the Android-based devices, which were used in the evaluation of the work in this thesis. One disadvantage of this kind of compass is the inability to output absolute values. Thus a calibration is needed to use the values given by the Hall elements.

However, regardless of the hardware used to implement the compass, the operational principles remain similar; the compass works by measuring the direction of Earth's magnetic field. The common way to obtain that direction is to measure the North-South and East-West factors of the field in a plane parallel to the surface of the earth. For a compass that is lying leveled, like in cars, it is sufficient to measure two separate axis-aligned magnetic components. If the compass is tilted, as frequently happens in a mobile device, the computed tilt should be compensated. This is achieved by using the a 3-axis accelerometer.

In practice, a compass may have some inaccuracy in the reported heading. There are three main sources for such inaccuracies:

- Magnetic field distortion: this is specially notable in urban environments where a lot of ferrous [1] materials and sources of electromagnetism are present. Some of those sources have fixed locations like rails and transformer boxes while others

1 iron, nickel and various compound materials that are used for permanent magnets for example

like cars or the magnets often encountered in handbags change in location. Any source of magnetism adds up to the total magnetic field observed by the sensor. Magnetic field vectors are added by standard vector addition. Therefore, local magnetic fields added to Earth's magnetic field can result in wrong heading information.

- Calibration errors: magnetic field sensors measure the magnetic field strength in two or three perpendicular directions. Since sensor elements behave differently and in most cases are only able to report values relative to the field strength, a calibration is needed. Typically the zero point has to be set for each sensor element. A bad calibration will result in offset values. The calculated orientation from these values can be different from the orientation of the observed magnetic field. Typically the error introduced by bad calibration is in the order of a few degrees.

- Internal noise: noise is inevitable for sensors. The amount of noise depends on the quality and resolution of the sensing elements. Unfortunately, for commercial reasons, sensors that are built into mobile devices are in general not of highest quality. The uncertainty caused by the sensor of the device used here is typically in the order of a few degrees.

2.2.3 *Accelerometer in Mobile Devices*

An accelerometer is needed to tell the tilt, i.e. the vertical direction

Accelerometers belong to the so-called Micro Elecro Mechanical Systems (MEMS). It measures the acceleration along its axis analogous to the mass-spring system. In such a system, the suspended mass M causes the length L of the spring to change by ΔL. By knowing ΔL and using Newton's Second Law[2], the induced acceleration can be computed.

The accelerometers can be differentiated according to the technique used to measure the change ΔL. For example, the *Asahi Kasei Microsystems AK8976A* found in Android devices uses a piezo-resistive accelerometer. This kind of accelerometers uses a piezo-resistive substrate and the force exerted by the seismic mass changes the resistance of the material. The resistance is translated then into acceleration.

As mentioned, an accelerometer can be used to compensate for the compass tilt. This is achieved by obtaining the pitch and roll angles from the accelerometer with reference to the local horizontal plane.

2 A body of mass M subject to a force F undergoes an acceleration A that has the same direction as the force and a magnitude that is directly proportional to the force and inversely proportional to the mass, i.e., $F = M \times A$

2.2.4 *GPS*

The Global Positioning System (GPS) is a satellite based system. The basic theory behind GPS is quite simple. To calculate a position, the distance to at least three satellites, which are stationary, is triangulated. For a more accurate result, the triangulation is done for all visible satellites and the result is averaged. In this thesis, the GPS coordinate system is used to reference the location of objects and points of interest. The GPS coordinate system uses the ploar coordinates (longitude, latitude, altitude) similar to standard spherical coordinates (r, θ, ϕ).

In general, GPS positioning is enough accurate for most of outdoor positioning purposes. However, GPS performance may degrade due to several factors like:

- Atmospheric effects: these typically cause the largest errors, also in open environments.
- Loss of LoS and multipath effects: in urban environments, the direct LoS required for reliable triangulation is often blocked by higher buildings. Additionally, the GPS signal gets reflected by buildings. This causes multipath effects which might lead to unresolvable triangulation problems.

2.2.5 *Mobile Device's Coordinate System*

The Android [1] mobile device used as a development platform in this thesis adopts a coordinate system as shown in Figure 4. Here, the origin is placed at the lower-left corner of the screen. The x-axis is pointing right, y-axis towards the top of the device and the z-axis points upwards away from the screen. The orientation sensor reports the rotation of the device relative to an earth-bound local East (x), North (y), Up (z) coordinate System. Therefore, heading in this coordinate system is the rotation around the z-axis. The rotation around x- and y- axes are termed *pitch* and *roll*, respectively. Pitch describes a nose up (up to $-90°$) or down (up to $90°$) rotation of the device where a value of zero denotes a leveled state. When pointing, this angle will be negative if one points upwards. This can be used, e.g., to determine if one is pointing towards a high object. The roll angle describes a rotation to the left or right. This is usually not of interest since roll's rotation axis is the same as the pointing vector, thus rolling does not change the direction of pointing.

2.2.6 *Level of Detail*

Level of Detail (LoD) techniques are usually used in computer graphics to reduce the complexity of 3D objects. In general, these

Figure 4: Android's coordinate system

technique aim at increasing the efficiency of rendering by decreasing the workload on the graphics pipeline stages, mainly vertex transformations. Such techniques have in general low impact on the object's appearance especially when the object is distant or moving fast [46]. Similar to applying a LoD to an object's geometry, LoD can be applied as well to manage an object's texture.

Considering the example object shown in Figure 5a-5b, the following interesting different kinds of LoD can be identified:

2.5D model can cause loss of details along the vertical axis

- *2.5D Model:* This is formed by extruding the object footprint by an amount that is equal to the object's height. The footprint itself is formed by taking a top-view projection of the 3D object on a horizontal plane. As shown in Figure 5c-5d, this LoD forms a good approximation of the object regarding the details in the horizontal plane. However, it can lead to distortion along the vertical axis, e.g. if the object has a triangular roof. The amount of data needed to store such a model depends on the complexity of the footprint. Assuming that N points are needed to describe the N vertices of the footprint, then a data structure with a total of 2N points is needed to describe the whole object (accounting for both the footprint and its extrusion).

Convex Hull model has no effect when the fingerprint has a rectangular shape

- *Convex Hull Model:* This is formed from a 2.5D model by firstly computing the *convex hull* of the footprint, where a convex hull is the smallest convex polygon containing all the vertices of the footprint [22]. The footprint is then extruded by the height of the object. It is out of the scope of this thesis to discuss the existing algorithms to find a convex hull (e.g. Divide-and-Conquer and Gift Wrapping [22]). As shown in Figure 5e-5f, the convex hull model tends to make the footprint (and hence the 3D object) lose some details (actually, it may make the footprint look bigger). However, as many buildings have rectangular footprints, a convex hull LoD will often have no influence on the final

shape of the object. Therefore, assuming that N points are needed to describe the N vertices of the footprint, then a data structure with a *maximum* of 2N points is needed to describe the whole object (accounting for both the footprint and its extrusion).

- *Axis-aligned Bounding Box Model:* This is formed from the 2.5D model by firstly computing the *axis-aligned bounding box (AABB)* of the footprint, where an AABB is established by finding the extreme vertices of the footprint in all three directions of the coordinate system [27]. Computing the AABB has relatively low computational complexity and it requires low and fixed amount of data to represent the object. Therefore, assuming that N points are needed to describe the N vertices of the footprint, then a data structure with exactly 2×4 points is needed to describe the whole object (accounting for both footprint and its extrusion). As shown in Figure 5g-5h, the main disadvantage of AABB is that it often causes the object to often look much bigger than its real size, especially if the footprint is not aligned along any of the axes.

AABB is memory-efficient but can dramatically change the modeled object

- *Oriented Bounding Box (OBB) Model:* This is formed from the 2.5D model by firstly computing the *oriented bounding box* OBB of the footprint, where an OBB is similar to an axis-aligned box in that it forms the tightest bounding box, but it differs from the axis aligned box in that it maintains the arbitrary orientation of the object in space as shown in Figure 5i-5j. While OBB has similar storage requirements as the axis-aligned box (i.e. exactly 2×4 points to describe the whole object), it is more computationally expensive.

OBB approximates the object much better than an axis-aligned box

2.2.7 *Viewer-centric Queries*

In this thesis, a *viewer-centric query* is defined as follows:

Definition 1. *A viewer-centric query is a spatial query that questions a spatial database and returns as a result one or more points of interest based on the viewer's current line-of-sight or field-of-view.*

LoS describes "an imaginary line from the eye to a perceived object´´. Consequently, a viewer-centric query based on the LoS has two functions:

- retrieving a POI associated to the object. In this thesis, this functionality is referred to as *point-to-find*. This query takes the LoS as an input parameter.

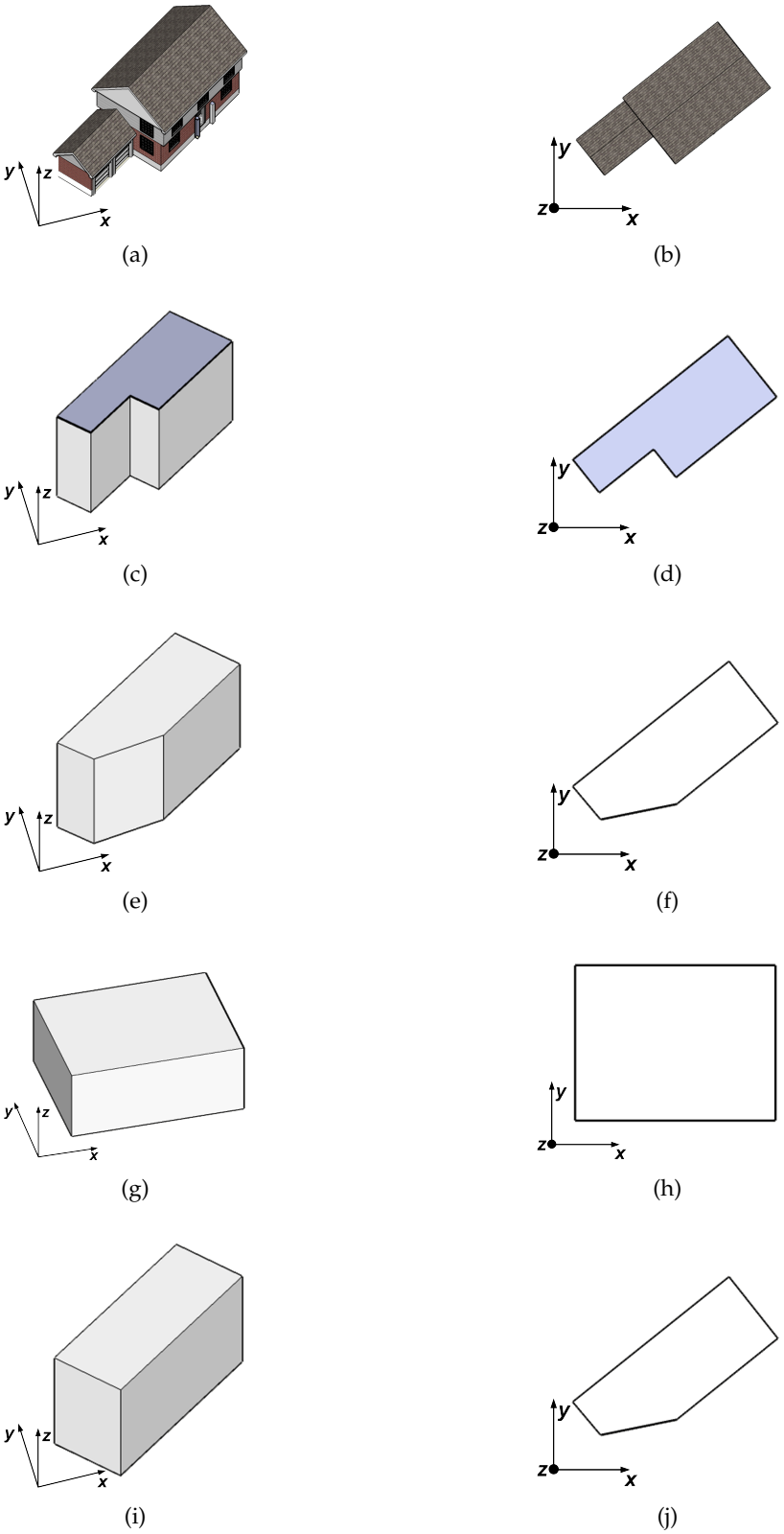


Figure 5: Different kinds of level of detail (left 3D view , right top view): (a)-(b) actual object, (c)-(d) 2.5D, (e)-(f) convex hull, (g)-(h) axis-aligned box, and (i)-(j) oriented box

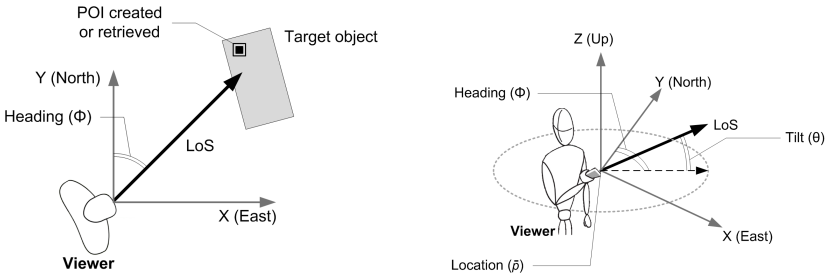


Figure 6: Modeling the user's LoS: top view (left) and 3D view (right)

- creating a POI and associate it to the object. In this thesis, this functionality is referred to as *point-to-tag*. This query takes both the LoS and the created POI as input parameters.

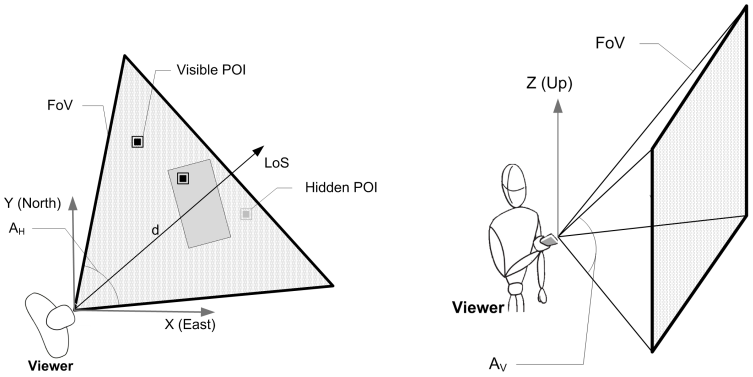
The location and orientation sensors embedded in the mobile device are used to model the viewer's LoS. As shown in Figure 6, the LoS is determined in a 3D space as a vector with the following parameters:

- the vector originates at the user's current location $\bar{p}$, which is measured from the location sensor (e.g. GPS),
- the heading of the vector (from north) is measured from the magnetic sensor (i.e. electronic compass), and
- the tilt of the vector (with respect to gravity) is measured from the accelerometer.

It is noteworthy here that the discrepancy between the modeled LoS (as a vector stemming from the mobile device) and the actual LoS (stemming from the viewer's eye) can have some influence on the accuracy of the viewer-centric queries. However, based on the investigations done in [57] and [30] this discrepancy is found to have less effect than the discrepancy caused , e.g. by GPS errors..

The FoV describes "the angular extent of the observable world that is seen from a given viewing point´´. The human FoV spans approximately 200° horizontally (considering both eyes) and 135° vertically [33]. Consequently, a viewer-centric query based on the FoV has two functions:

- retrieving all POIs lying within the FoV. In this thesis, this functionality is referred to as *point-to-browse*. This query takes the FoV as an input parameter.
- retrieving all POIs lying within the FoV and are matching a given search criterion (e.g. category of POI). In this thesis, this functionality is referred to as *point-to-search*. This query takes both the FoV and the search criterion as input parameters.



(a) (b)

Figure 7: Modeling the user's FoV: top view (left) and 3D view (right)

As shown in Figure 7, the FoV itself is modeled as a pyramid-shaped frustum with a rectangular base such that:

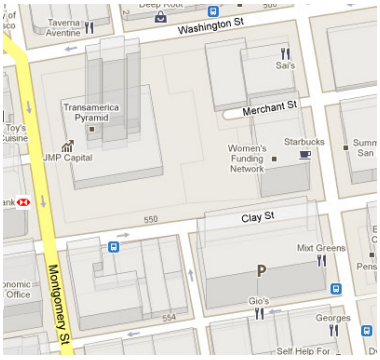
- the head of the pyramid lies at the user's current location, roughly elevated by the height of the user,
- the axis of the pyramid aligns with the LoS (determined in the same way done for the LoS-based queries i.e. from location and orientation sensors),
- the height of the pyramid equals the viewing range of the user d,
- the angle between the vertical triangular faces equals the user's horizontal viewing angle A_H, and
- the angle between the horizontal triangular faces equals the user's vertical viewing angle A_V.

As shown in Figure 7, in real world, besides the user's viewing direction and viewing range, the geometry of objects existing in the user's surrounding plays a major role in deciding the visibility of POIs from the user's current location. The following section describes the kind of geometry models being used by the approach proposed in this thesis.

2.2.8 *World Geometry Models*

The work in this thesis assumes availability of information about objects (e.g. buildings) along the height dimension. The obvious method to achieve this information is to apply 3D geometrical modeling (e.g. in Figure 8a). However, so far the cost and effort to produce such models for large areas have been prohibitive. Moreover, the high level of detail provided by such models will be over kill for the approach proposed by this thesis, where only the

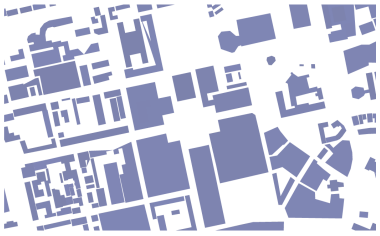
(a) Full 3D model (from Google Earth)



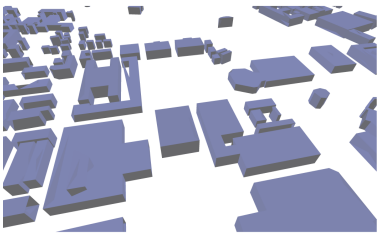
(b) 2.5D model in Google Maps

general shape of the object is of interest and not the other details, e.g. about textures and colors (as will be explained in Chapter 3). Therefore, this thesis adopts a 2.5D geometrical modeling as defined in Section 2.2.6.

The advancements in aerial imagery have reduced the cost for building 2.5D models at large scale [50]. Google Maps e.g. started offering a 2.5D building modeling (see Figure 8b). Unfortunately, this view is offered only for some cities in the USA and besides so far there is no public API to access this data. Therefore, to afford such information, OpenStreetmap[3] offers a good alternative. OpenStreetmap is an open community-based mapping project. Although no 2.5D data is offered here, 2D building floor plans (or footprints) are available for many places. The fingerprints can be exported as rendered images and as XML structure. Figure 8a shows some of the fingerprints in the city of Darmstadt, Germany. Appendix B gives the corresponding XML description for the depicted fingerprints.



(a) OpenStreetMap's 2D fingerprints



(b) 2.5D model extruded from the fingerprints

After exporting the fingerprints, they need to be extruded to the heights of the buildings so that a complete 2.5D model is constructed. For the work in this thesis, the heights of the buildings in the test areas have been surveyed manually using a LASER distance meter. Figure 8b shows the resulting 2.5D model

3 http://www.openstreetmap.org/

for the fingerprints in Figure 8a. The 2.5D models can be stored in any spatial database, e.g. MySQL with Spatial Extensions [4], which supports standard geometry operators (e.g. a window operator to retrieve a subset of the stored 2.5D model).

3

A FRAMEWORK AND A QUERY MODEL FOR VIEWER-CENTRIC MOBILE SERVICES

This first part of this chapter provides an overview of iVu.KOM; the proposed framework for viewer-centric mobile services. The different components of the architecture and their functions are explained. The second part introduces the iVu model as a central element of the proposed framework. In specific, it describes in further detail the functions of the components that are responsible for generating the iVu model. Design decisions taken to support query efficiency are explained in detail.

3.1 THE IVU.KOM FRAMEWORK

iVu.KOM framework is proposed as a framework for viewer-centric mobile services with focus on achieving the following main goals:

iVu.KOM solves the drawbacks of the other approaches

- Enhancing the query efficiency,
- enhancing the query accuracy,
- facilitating the user's awareness of the achieved query accuracy, and
- supporting the retrieval as well as the creation of POI's (user-generated content).

To achieve these goals, iVu.KOM is basically based on state of the art design of client-server architectures. Recently, 3-tier client-server architecture has been applied as a common practice for building scalable and modular distributed applications [59]. As shown in Figure 8, this architecture is formed of the presentation tier, the logic tier and the data tier. Most commonly run on the client, the presentation tier encompasses the user interface allowing the user to interact with the application. The logic tier offers different business objects, i.e. working units that control an application's functionality by performing detailed processing. The data tier holds the different data (normally in a database) on which the application works. Both the logic and data tiers are physically running in the server domain.

iVu.KOM is designed with a 3-tier architecture in mind, however still having the possibility to introduce its own extensions on the client side to enable LBS applications working on top of iVu.KOM to seamlessly run viewer-centric queries. As shown in

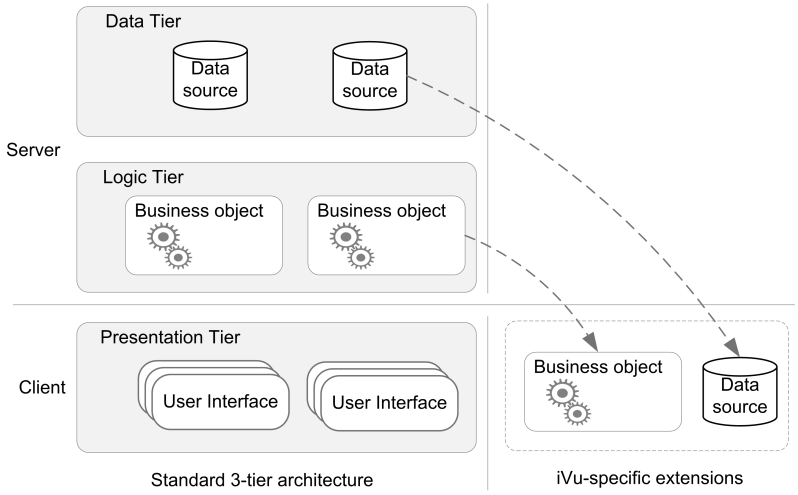


Figure 8: Standard 3-tier architecture (left) with iVu-specific extensions (right)

the right part of Figure 8, iVu.KOM allows a part of the business logic and the application data to be shifted to the client. In this respect, iVu.KOM can be functionally analogous to the Asynchronous JavaScript and XML (AJAX) which enables part of the logic (e.g. updating the user interface) to be executed on the client side, therefore enabling a richer and more responsive user experience.

Analogous to the AJAX paradigm, iVu.KOM functionality spans the client and the server

Figure 9 depicts the internal architecture of iVu.KOM. The arrows in the figure show the data flow between adjacent components, where each component produces the kind of information within the parentheses. In the following is an overview of the different client and server components:

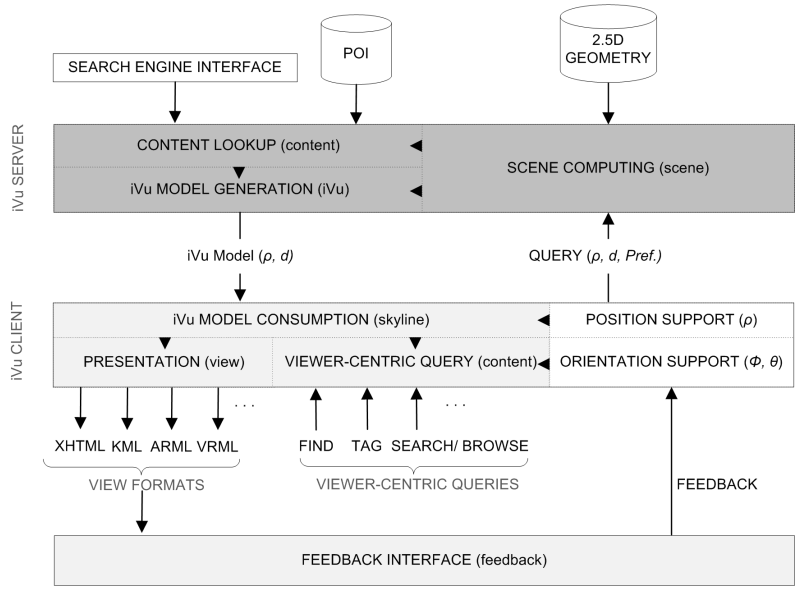


Figure 9: Architecture of the iVu.KOM framework

3.1.1 *Scene Computing Component*

The scene computing component runs on the server and handles the process of selecting a sub-set of the geometry information maintained in the 2.5D spatial database (see Chapter 2) according to different query parameters like the user's location ρ and the scene size d. Section 3.2 describes in depth how this component retrieves the scene and applies geometry simplification in the form of oriented bounding boxes. Therefore, this component is crucial in supporting the requirements of query promptness and efficiency as stated in Chapter 1.

3.1.2 *Content Lookup Component*

The content lookup component runs on the server. It receives the scene from the scene computing component and retrieves geo-tagged content that lies within the scene and matches the search criteria, if specified by the user. Geotagged content can come from a dedicated POI database where POI are already categorized or it can be a search result using a geo-enabled search service like Google Maps [10] and Flickr [9].

3.1.3 *iVu Model Generation Component*

The iVu model generation component runs on the server. As input parameters, this component accepts the scene model created by the scene computing component, and the geotagged content (i.e. POIs). As output, this component gives the iVu model. section 3.2 explains the design decisions behind the iVu model in order to to align with the requirements of query promptness and efficiency.

3.1.4 *iVu Model Consumption Component*

The iVu model consumption component runs on the client. It accepts the iVu model and the current position $\bar{\rho}$ of the user to compute a *skyline view*. A skyline represents a 360° panoramic view of the surrounding geometry as it appears from the current position (or viewing point) of the user. Skyline proves very useful for several purposes. Firstly, it serves as a basis for executing the viewer-centric queries directly on the client as will be shown in Chapter 4. Secondly, skyline is very helpful in understanding and modeling of the sensor uncertainties and their effects on the spatial queries as shown in Chapter 4. Last but not least, Chapter 5 shows how the skyline can be utilized to make the user aware of the achieved query accuracy and to react to quality degradation.

Each component of iVu.KOM contributes to query promptness and query accuracy

3.1.5 *Viewer-centric Query Component*

Running in the client, the viewer-centric query component accepts the skyline view and the readings of the GPS receiver and the orientation sensors: the heading ϕ from the compass, and the tilt θ from the accelerometer. Chapter 4 shows how this component supports execution of different viewer-centric queries like the point-to-find/tag and point-to-search/browse queries. This component allows integration of the user's feedback to improve the query accuracy. Therefore, this component is equally important for supporting both query accuracy and query promptness.

3.1.6 *Presentation Component*

Running on the client, the presentation component transforms the skyline view into an application-specific format. Therefore allowing same information to be presented according to the needs of the application or the user interface, e.g. XHTML for viewing in a classical web browser and Augmented Reality Markup Language (ARML) [16] for augmented reality viewing. Chapter 5 provides a design of an Augmented Reality interface for the iVu.KOM framework.

3.1.7 *Position Support Component*

The depicted positioning support component is not part of the framework. iVu.KOM can interface with any location provider including device-centric systems like GPS and network-centric systems, e.g. based on GSM cell-ID. Chapter 6 shows the design and implementation of *WBroximity*, a WLAN and Bluetooth-based positioning service that can be used as a location provider for iVu.KOM, specially in indoor environments.

3.1.8 *Feedback Interface*

The feedback interface has two main functions: displaying information about the query accuracy and enabling the user to provide feedback about the achieved query accuracy. The feedback interface displays the information about accuracy in a form suitable to the current presentation format. For example, Chapter 5 suggests to visualize the query accuracy for the augmented reality interface by the amount of mismatch between the computed skyline and the underlying camera image.

The iVu model is a representation produced by the iVu server in response to a query request QUERY(ρ, d, Pref.), where ρ is the user's location when the query was issued, d is the scene radius, and Pref. is the search criteria or preferences[1].Unlike queries in conventional geospatial search systems which return only a set of geo-referenced content as a result, the iVu model contains two kinds of information: *scene* and *POI's* information. The scene is a description of the geometry in the area surrounding the user's location ρ. The POI's are the collection of geo-tagged information within the scene area. In this respect, the iVu model conveys to the client all the information needed to execute viewer-centric queries (e.g. point-to-find and point-to-search) locally on the client.

3.2.1 *Scene Computing*

Different strategies are applied to generate an efficient scene

In favor of supporting efficient query execution on the client, it is crucial that the scene included in the iVu model is to be represented in a way that is efficient for computing. For this purpose, two main steps are applied by the iVu.KOM framework during the scene computing: limiting the radius of the scene and simplifying the scene representation.

Step 1: Limiting the Scene Radius: From an incoming query request QUERY(ρ, d, Pref.), the scene computing component uses the location ρ and scene radius d to retrieve the geometry around ρ. Therefore, this step involves running a standard bounding box query on the 2.5D geometry dataset as shown in Figure 10a. This results in a subset of the geometry that lies within or intersects with the area defined by the query box as shown in Figure 10b.

(a) (b)

Figure 10: (a) Range query (b) Query result

Step 2: Simplifying the Scene Representation: After limiting the scene radius and getting the enclosed geometry, the scene representation is simplified. This means that the geometry is described with less data, which in turn has several advantages, including:

1 The Pref. parameter is used in the case of a point-to-search query

- smaller scene size. This makes it more efficient to transmit the scene to the client, in terms of network bandwidth and transmission delay,
- less complex client-side processing, and
- lower client-side memory footprint.

A concept that is closely related is the *Level-of-Detail) (LOD)*. LoD techniques are usually used in computer graphics to reduce the complexity of 3D objects. In general, these technique aim at increasing the efficiency of rendering by decreasing the workload on the graphics pipeline stages, mainly vertex transformations. Such techniques have in general low impact on the object's appearance especially when the object is distant or moving fast [46]. Similar to applying a LoD to an object's geometry, it can be applied as well to manage an object's texture.

Effective size defines how big an object in the FoV is

In the case of viewer-centric queries, it is irrelevant how the object is textured or colored. In fact, it is mainly how big an object appears in the FoV of the user that plays much role. It is referred to this as *the effective size of the object*. A bigger object will typically occupy larger portion of the FoV, and hence, a point-to-tag query for example will have higher chance to match that object. The effective size depends on the following three factors:

- the actual size of the object,
- the distance to the object, and
- the orientation of the object.

The maximum horizontal and vertical angles that an object spans in the FoV from a given viewing point are adopted as a quantitative measure for the effective size of the object, . For simplicity, Figure 11 explains the horizontal angle θ depending on the aforementioned factors. As shown in Figure 11a, the object occupies an angle θ_1 at distance d_1. Figure 11b shows the same object has $\theta_2 < \theta_1$ when it is at a distance $d_2 > d_1$. Figure 11c shows that the object has even $\theta_3 < \theta_2$ at the same distance, i.e. $d_3 = d_2$, however the object has different orientation with respect to the viewing point, which makes it look smaller.

A Level-of-detail for viewer-centric queries should ideally respect the object's effective size

From the above argument, it is essential for the purposes of viewer-centric queries to apply a level-of-detail that does not dramatically distort the effective size of the object. This is important to increase query efficiency while maintaining a good query accuracy.

Based on the analysis in Chapter 2 of the different kinds of LoD, a decision was made to adopt an OBB model for the objects in the iVu scene. The justification for this decision is three-fold:

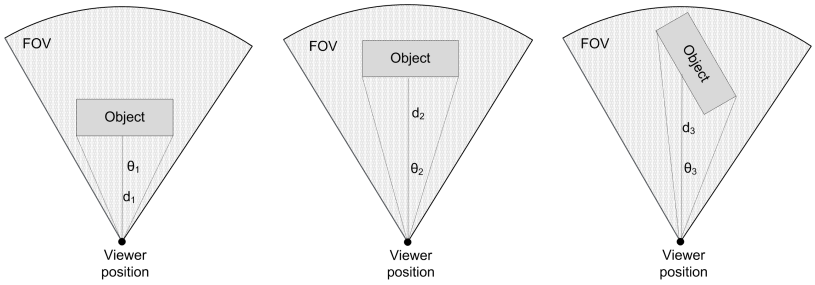


Figure 11: Object's effective size affected by (a) object's dimensions (b) distance to object and (c) object's orientation

- An OBB model holds a sufficient LoD needed to run viewer-centric queries without loosing accuracy while having compact geometry description. As shown in Figure 5, OBB approximates the original shape and alignment of the actual object with minimal storage requirements. Both the 2.5D and convex hull models provide good approximation of the actual geometry but they require more points to describe the object. Finally, the *axis aligned bounding box* uses the same amount of data as the OBB to describe the object, but it can generally cause severe distortion to the object's effective size.

- An OBB uses a fixed amount of data to describe an object regardless of the geometry complexity. This means that the size of the scene will be proportional to the number (or density) of objects within that scene. This is useful for automatically adjusting the scene radius according to the type of the environment. For example, in rural and open environments the radius may be set to a large value, while in urban and dense environments the radius can be set to smaller values.

- It is true that an OBB is more computationally expensive than an axis aligned bounding box or convex hull. However, according to the iVu.KOM framework, such computations are shifted completely to the server and do not count on or impact the processing capabilities of the client.

3.2.2 *POI Inclusion*

POIs represent the content in the query response

POIs represent the useful content that the user wishes to get in the result of a query. For iVu.KOM, any geo-referenced content like geo-tagged photos, blogs, web pages etc. can be used as a POI. A POI is represented by the following pieces of information:

- An anchor point $a(x, y, z)$, where the x, y and z are the longitude, latitude and altitude GPS coordinates, respectively. An anchor point specifies where a POI is placed in the physical world.
- A short textual description of the content associated with that POI. This is useful to give the user a fast hint about the type of the POI and also for filtering and categorizing the POI's on the client side.
- A Uniform Resource Locator (URL) that uniquely specifies where the content can be found in the digital world, i.e. the internet.

This POI model allows considering the POIs in the viewer-centric queries as the physical location of each POI is already included in the iVu model. Additionally, including the URL allows to get further information about any POI that appears in the query result. Such information can be then presented according to the type of the POI. For example, a web page link can be opened in the browser and a video may be played in a video player like Youtube.

Different filters are applied to POIs in iVu.KOM

The $Pref.$ parameter included in the $QUERY(p, d, Pref.)$ can be used to increase the relevance of the returned POIs. This parameter is used when querying the POI repository or searching the search engine. Therefore, the Content Lookup component (see Figure 9) can basically apply two filters to the searched POIs:

- The scene filter. This causes to include only POIs whose anchor points lie within the scene area. A simple point-window intersection will achieve this filter functionality.
- The preferences filter. A preference can be specified on the business category of the POI to be retrieved (e.g. restaurants, hospitals etc.). It can be also specified on the content type of the POI, e.g. only web pages, videos etc. To achieve this filter, POIs in the POI database are marked by their business category and their content type (e.g. as MIME content-type [31]). When no preferences are specified, the POIs are retrieved in browsing mode, i.e. all POIs that pass the scene filter will be included in the iVu model.

As will be shown in Chapter 4, the viewer-centric queries that are run on the client, effectively perform as a third filter which retrieves only POIs lying within the FoV.

3.3 IVU MODEL GENERATION

The preceding sections showed how to construct the two elements contained in the iVu model: the scene and the associated

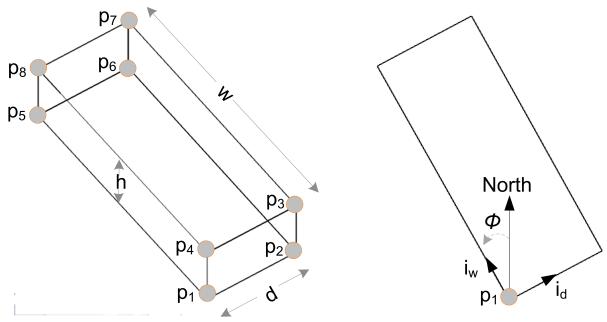


Figure 12: Representing an OBB

POIs. To serialize this data to the client, it is necessary to find a relevant representation and a relevant expression format for this representation.

3.3.1 *iVu Model Representation*

An OBB representation should have a low memory footprint

As far as the scene is concerned, there are many possible ways to represent an OBB. With reference to Figure 12, Table 1 summarizes some of these possibilities. The first representation is the straightforward way to represent an OBB. It uses a construct of 8 vertices, where each vertex is specified as a point in a 3-axis coordinate system using the GPS latitude, longitude and altitude coordinates. Assuming that a GPS coordinate is specified in radians as a single-precision IEEE float (i.e. with size of 4 bytes), then a total of 96 bytes memory will be needed to hold each OBB. The second representation makes a memory saving by storing only 4 vertices (of the base or footprint) and the height of the object. Assuming the same GPS representation as above, then an OBB takes up 52 bytes memory. The disadvantage here is that the client needs to perform 4 sum operations to reconstruct the full OBB. The third representation uses only one vertex, the unit vectors (i_w, i_d, i_h) representing the orientation of the object local axes, and the object's dimensions (w, d, h) along the local axes. This representation needs 48 bytes memory. The last representation is similar to the third one, but it specifies the orientation using the heading angle ϕ of object main axis (the $w-$axis) from north. This representation needs only 28 bytes memory (about 70.8% memory saving compared to the 8-vertex construct), therefore the OBB representation is adopted in the iVu model.

With respect to the POI representation, a construct containing the anchor point (amounts to 12 bytes when using radian GPS coordinates in IEEE single precision), the POI description (varying byte size) and the POI URL (varying byte size) is needed.

3.3.2 *iVu Model Format*

iVu format should be compact and simple to process

A geography markup format or language is needed to express the aforementioned constructs. This format should be ideally compact which leads to have as much bandwidth and memory savings as possible. The format should also be simple to process by the client which leads to responsive query execution and battery savings. The existing geography markup languages have some drawbacks in this respect. On one hand, some of these formats are designed for specific output style. For instance, KML [6] is intended for display on a map like Google Maps. Therefore, expressing the iVu model with KML will lead to unneeded additional processing steps, as the iVu model is not basically intended for direct display on a map (as shown in Chapter 4). On the other hand, other formats like GML [36] have a verbose structure when describing geometric shapes, which breaks our compactness requirements.

Construct Elements	Memory Usage (Byte)
$\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$	96
$\{p_1, p_2, p_5, p_6, h\}$	52
$\{p_1, i_w, i_d, i_h, w, d, h\}$	48
$\{p_1, \phi, w, d, h\}$	28

Table 1: Some OBB representation constructs

For these reasons, the iVu model uses a simple XML format to express its semantics. Listing 1 shows a snippet of an iVu model. The root element **ivu** has a **ts** attribute specifying when the iVu model was generated, which is useful when applying time-based caching policies. The **ivu** element has as well a **lat**, **lng** and **alt** attributes specifying respectively the latitude, longitude and altitude of the user's location when the $\mathrm{QUERY}(\rho, d, \mathrm{Pref.})$ was issued. This is useful when applying location-based caching policies. Within the **ivu** element, a set of **obj** elements can appear. An **obj** element describes an OBB within the scene using its **lat**, **lng**, **alt**, **hdg** (heading), **w** (width), **h** (height) and **d** (depth). The **hdg** is specified as an angle in the range $[0, \pm\pi]$, where a positive **hdg** mean that the width axis is aligned to the east of the north, and a negative **hdg** means that the axis has an alignment to the west of the north. The **poi** element may appear as a direct child of an **obj** element. In this case, it is optional to specify the absolute location of the POI, where the POI will appear in any query that matches the parent object. Alternatively, the **poi** can appear as a direct child of the **ivu** element. In this case, the absolute location of the POI is to be specified. Analogous to hyperlinks on classical web

pages, the **href** attribute of **poi** specifies the URL of the content referred by that POI. The **descr** provides a short description of the POI. This description is included in the result of the viewer-centric queries that are run on the client.

Listing 1: Format of the iVu model

```
<ivu ts="2010-07-28-11-37-42" lat="49.874671" lng="8.660655"
    alt="140"/>

 <obj name="Main Building" id="obj-1">
  <obb lat="49.874681" lng="8.660666" alt="145" hdg="81.08" w
      ="59.21" h="15" d="21.7" />
        <poi id="poi-1" href="www.ivu.main-building.de" desc=
            "iVu Workshop" />
 </obj>

 <poi id="poi-2" href="www.tu-darmstadt.de" descr="TUD" lat="
     49.874562" lng="8.660633" alt="176"/>

</ivu>
```

4

THE SKYLINE AND VIEWER-CENTRIC QUERIES

This chapter presents the functions of the components highlighted in Figure 13. The chapter starts by describing the concept of the skyline and its derivation from the iVu model. The last part explains the usage of the skyline as a means for efficient and prompt execution of viewer-centric queries on the client.

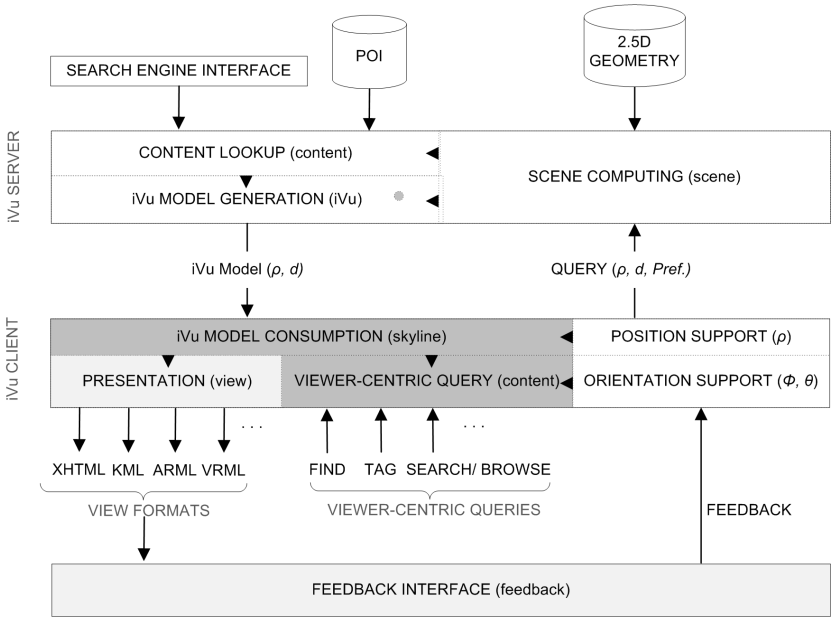


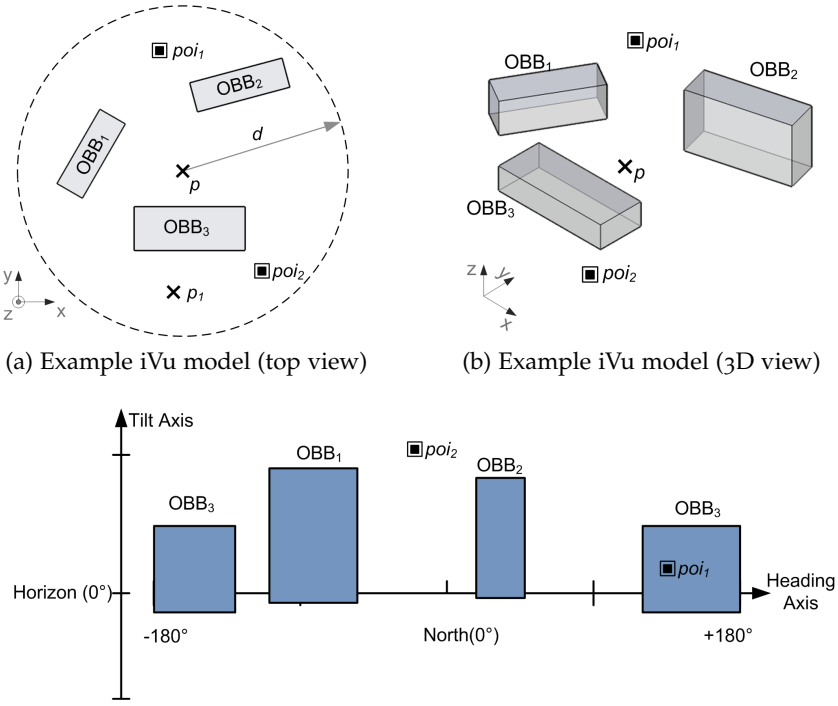
Figure 13: Components covered by this chapter

4.1 THE SKYLINE CONCEPT

As mentioned in Chapter 3, as a response to the QUERY$(\rho, d, \text{Pref.})$, the client receives an iVu model iVu(ρ, d) which describes the scene and the POIs within a radius d around the user's location ρ. With the availability of this information on the client together with the heading, tilt and position information that are read from the client's sensors, it is possible now to perform viewer-centric queries directly on the client. The *skyline* serves as a basis to resolve viewer-centric queries. Technically, the skyline is a projection of the iVu model on a sphere centered at the current position of the user. Therefore, the skyline can be seen as a 360° panoramic view of the geometry and the POIs contained in the iVu model.

The skyline provides a 360° panoramic view of the iVu model

The concept of the skyline is best explained with an example. In Figure 14 it is assumed that the client obtained an iVu model



(a) Example iVu model (top view)

(b) Example iVu model (3D view)

(c) Corresponding skyline at $\bar{p} = p$

Figure 14: Skyline concept

The skyline is specified in angular units along a heading and tilt axes

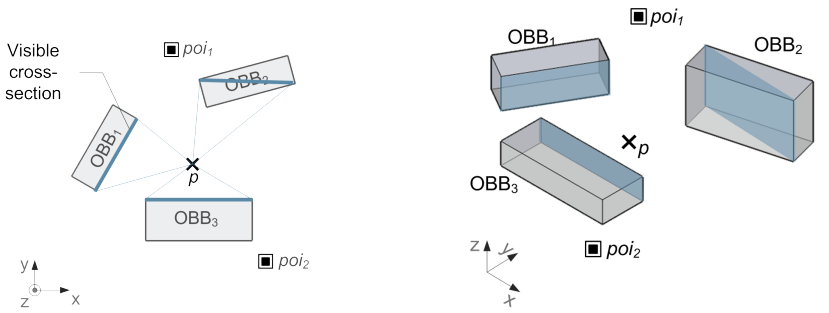
with a scene composed of 3 bounding boxes ('OBB1', 'OBB2' and 'OBB3') and 2 POIs (poi_1 and poi_2). Assuming that the user is positioned at point $\bar{p} = p$, the skyline will be as shown in Figure 14c. Effectively, the skyline shows how each element (bounding box or POI) of the iVu model looks like in the user's field-of-view from the current position. Therefore, and as a normal extension to the *effective size* concept explained in Chapter 3, the skyline is specified in angular units in a 2-axis system where the horizontal axis is the *heading axis* and the vertical axis is the *tilt axis*. The following explains how a skyline is derived from the iVu model.

4.1.1 *Projecting the Bounding Boxes*

Bounding boxes in the iVu model are mapped into billboards in the skyline

Each bounding box in the iVu model is mapped into a *billboard* in the skyline. A billboard is a rectangle which is computed by firstly finding the visible cross-section of the bounding box. This task, because a bounding box in the iVu model uses an OBB-based model, reduces to a case-selection among three options: the width side of an OBB is visible (as for 'OBB1' and 'OBB3'), the depth side is visible, or the diagonal cross-section is visible (as for 'OBB2') as shown in Figure 15.

After finding the visible cross-section, the corresponding billboard is specified by the following parameters:



(a) Visible cross-sections (top view)

(b) Visible cross-sections (3D view)

Figure 15: Finding the visible-cross section of each bounding box

Angular parameters of a billboard

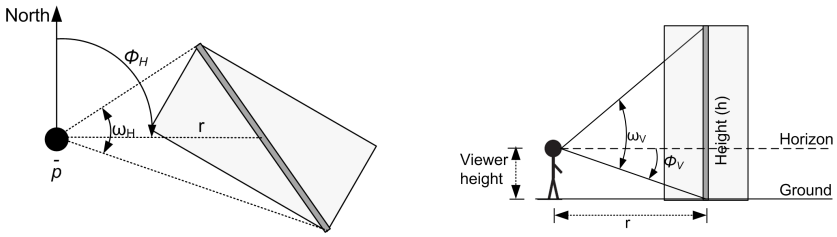
- The horizontal angular position ϕ_H of the billboard. This designates the location of the billboard on the heading axis of the skyline and is measured as the heading angle of the line connecting $\bar{\rho}$ and the center point of the cross-section, as shown in Figure 16a. As a convention, $\phi_H = 0°$ for objects located exactly along the north, $0° < \phi_H \leqslant 180°$ for objects located east of north, and $-180° < \phi_H \leqslant 0°$ for objects located west of north. To align with the 360° range of the heading angle, objects near the north or south are wrapped along the heading axis, as for 'OBB3' in Figure 14c.

- The horizontal angular dimension ω_H of the billboard. This designates the angular width of the billboard, i.e. the range it occupies on the heading axis of the skyline and is measured as the horizontal angle between the left and the right sides of the cross-section, as shown in Figure 16a.

- The vertical angular position ϕ_V of the billboard. This designates the location of the billboard on the tilt axis of the skyline and is measured as the vertical angle between the horizon and the line connecting the viewing point and the bottom of the cross-section, as shown in Figure 16b. As a convention, $\phi_V = 0°$ for objects located exactly on the horizon, $0° < \phi_V \leqslant 90°$ for objects above the horizon, and $-90° < \phi_V \leqslant 0°$ for objects located below the horizon.

- The vertical angular dimension ω_V of the billboard. This designates the angular height of the billboard, i.e. the range it occupies on the tilt axis of the skyline and is measured as the vertical angle between the top and the bottom sides of the cross-section, as shown in Figure 16b.

- The distance to the billboard r, measured as the distance between $\bar{\rho}$ and the center point of the cross-section. This parameter is not represented directly on the skyline but



(a) Horizontal dimension and placement

(b) Vertical dimension and placement

Figure 16: Parameters of a billboard

plays a role when performing the visibility computation as will be explained later.

4.1.2 *Projecting the POIs*

A POI in the iVu model is mapped into a point in the skyline

Each POI in the iVu model is mapped into a point in the skyline. The point has the following parameters:

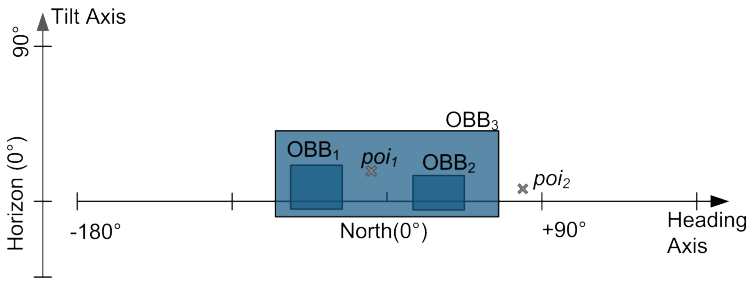
- The horizontal angular position ϕ_H of the POI. This designates the heading axis coordinate of the POI and is measured as the heading angle of the line connecting $\bar{p}$ and the POI. Similar to the billboard, this angle lies in the range $(-180°, 180°)$.
- The vertical angular position ϕ_V of the POI. This designates tilt axis coordinate of POI and is measured as the vertical angle between the horizon and the line connecting the viewing point and POI. Similar to the billboard, this angle lies in the range $(-90°, 90°)$.
- The distance r to the POI, measured as the distance between $\bar{p}$ and the POI. This parameter is not represented directly on the skyline but plays a role when performing the visibility computation as will be explain later.

4.1.3 *Characteristics of the Skyline*

Having explained the concept of the skyline, the following characteristics can be identified:

The skyline is location-dependent

- The skyline is location-dependent: even for a short user movement, the projected skyline may change considerably. For billboards, this change generally affects both the effective size of the billboards as well as their position on the heading and tilt axes of the skyline. Similarly, the POIs change their tilt and heading. This sensitivity to the viewer's



(a)

Figure 17: The skyline at position $\bar{\rho} = \rho_1$

location supports the choice of the skyline as a basis to resolve viewer-centric queries. Figure 17 shows the skyline computed at position $\bar{\rho} = \rho_1$ (marked in Figure 14)

- The skyline can be re-computed on the client instantaneously and without need to contact the server, as long as the client is moving within the validity range of the iVu model, i.e. for each $\bar{\rho}$, where $\|\bar{\rho} - \rho\| \leqslant d$. Beyond this range, the client needs to obtain a new iVu model [1]. This feature of the skyline leads to running queries in a promptly manner.

The skyline can be re-created on the client

- The skyline is independent of the tilt and heading of the mobile device itself. . These two pieces of information affect only the LoS and FoV which are projected on the skyline, as explained in the next section.

The skyline is independent of the tilt and heading sensor readings

4.2 USING THE SKYLINE FOR VIEWER-CENTRIC QUERIES

With its location-dependent description of the surrounding environment, the skyline provides a direct means to run viewer-centric queries, including the point-to-tag/find and point-to-search/browse queries, as explained next.

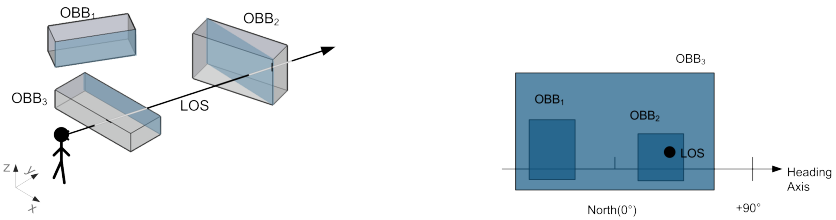
4.2.1 *Point-to-tag/find Queries*

A point-to-tag/find query involves as a first step identifying the target object being pointed to, i.e. the first visible object which is intersecting with the LoS. The LoS is determined as described in Chapter 2.

The nature of the skyline makes finding the target object of a pointing query simple and computationally efficient. The algorithm for this purpose can be summarized in the following steps:

A point-to-tag/find query is resolved by projecting the LoS as a point in the skyline

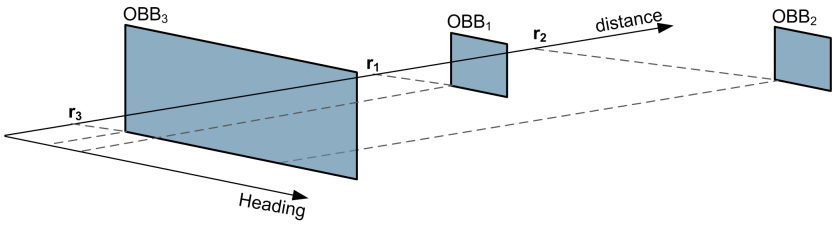
1 Assuming no caching of the iVu model is applied.



(a) Point-to-tag/find query (3D view)

(b) LoS projected on the skyline

Figure 18: Resolving a point-to-tag/find query with the skyline



(a)

Figure 19: Sorting the billboards with respect to their distance from the user's position

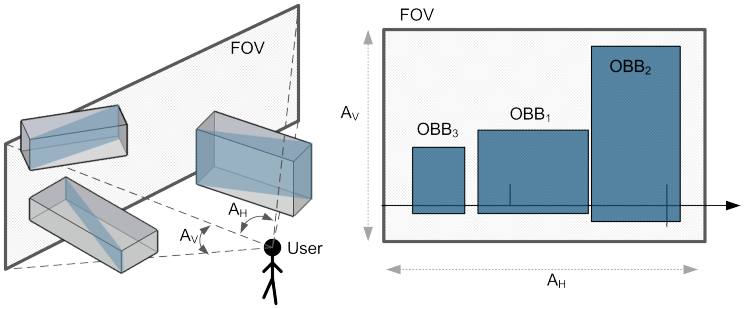
- the LoS vector is projected into the skyline as a point whose coordinates correspond respectively to the tilt and heading angles of the LoS vector, as shown in Figure 18.
- if the projected point lies *inside* only one billboard, then that billboard is taken as the target object. The query result will be then the POI associated with that billboard.
- if the projected point lies inside more than one overlapping billboard as shown in Figure 18, then the billboards are r-sorted, i.e. they are sorted with respect to their distance r from the user's current position $\bar{\rho}$. Then the billboard with the shortest distance is taken as the target object. Figure 19 shows the r-sorting of the the billboards in Figure 18. Here $r_3 < r_1 < r_2$, therefore 'OBB3' is identified as the target.
- if the projected point does not lie within any billboard, then the query result is empty.

4.2.2 *Point-to-search/browse Queries*

A point-to-search/browse query aims at identifying a set of visible POIs or visible objects within the FoV of the user, where the FoV is determined as described in Chapter 2.

Similar to projecting the LoS on the skyline, the first step in resolving the point-to-search/browse queries is to project the FoV on the skyline. Figure 20 shows that the FoV can be projected as

A point-to-search/browse is resolved by projecting the FoV as a window in the skyline



(a)

Figure 20: Resolving a point-to-search/browse query with the skyline

a window in the skyline, where the width and the height of the window are equal to the horizontal and vertical viewing angles, respectively. The window is placed such that it is centered around the projection of the LoS vector.

With the FoV projected into the skyline, two cases can be differentiated when evaluating the point-to-search/browse query:

1. *Finding visible POIs:* in this case, the query is supposed to return a set of visible POIs within the FoV. As a first step, all POIs lying outside the FoV are ignored. For POIs inside the FoV, a POI is considered visible if it is not *inside* a billboard. If the POI is contained within one billboard or more, the billboards are r-sorted as explained in the previous section, then the POI is considered visible if its distance from the $\bar{\rho}$ is less than the least distance to any billboard.

Excluding the POIs outside the FoV makes the query more efficient

2. *Finding visible objects:* in this case, the query is supposed to return information about a set of visible objects. As a first step, all billboards that are completely lying outside the FoV are ignored. Billboards that lie partly or completely inside the FoV are checked for visibility. In general, a billboard is visible if its top side (or part of it) is visible. If the top side of a billboard is hidden, then the billboard is commonly hidden. This means it is only necessary to track the heights of the billboards during the test. This is achieved by using an array of 360 float values; one for each degree along the heading axis. Each value represents the maximum billboard height so far encountered along that heading. Each billboard top is tested against the height array and if it is higher than the current maximum height then this billboard is visible and the maximum height is updated. This test has a resolution of one degree horizontally. However, the resolution can be easily increased by doing the same more than once per degree. This test scales linearly with the number of billboards. Each additional billboard requires as much additional tests as wide it is (in degrees). It is worthy to

Excluding the billboards outside the FoV makes the query more efficient

note that the billboards which are further away are smaller in size, therefore the test gets faster towards the end.

Filters applied by iVu.KOM to increase the relevance of a query result

To summarize, a viewer-centric query can provide the user by most relevant information to the current context just by pointing (or *looking*) at different targets in the surrounding. Effectively, the following filters are being applied during the query process to increase the relevance of the returned information:

1. The scene radius d. This limits the objects and POIs in the iVu model to a specific range that is typically reachable by a pedestrian user. The radius can be set as well to the maximum viewing distance of a human user.
2. The LoS or FoV. This confines the query further to objects or POIs that are lying along the LoS or within the FoV. On one hand, this means that only information in the actual viewing direction is being considered. On the other hand, this increases the query efficiency as the number of POIs or billboards to test is considerably reduced.
3. The billboards (or geometry). The billboards included in the visibility test provide also a natural filter that blocks POIs or other billboards which are hidden from the user.
4. The search preferences of the user. If the user specifies a search preference (e.g. a category of POIs), then only POIs matching this preference are returned.

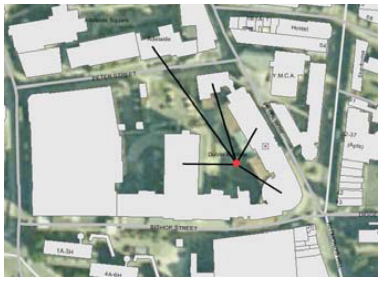
4.3 COMPARISON TO RELATED WORK

The following sections spot how iVu.KOM differs from major works in the area of viewer-centric services.

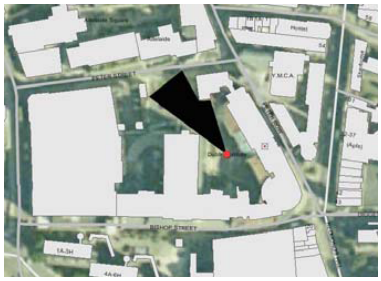
4.3.1 *The Work of Gardiner et. al.*

The approach proposed by Gardiner et. al. [33] can be summarized as follows:

- the client on the mobile device reads the current location ρ, heading ϕ and tilt θ and use them to build a directional spatial query.
- the query is sent to the server for processing.
- the server executes the query against a spatial database containing a 3D block geometry model. The query can be executed as a *vector query* which retrieves the POI associated with the object intersecting with the LoS (as shown in Figure



(a) A vector (LoS) query



(b) A frustum (FoV) query

Figure 21: Server-side directional queries againts spatial database (source [33])

21a). Alternatively, the query can be executed as a *frustum query* which retrieves all POIs within the users FoV (as shown in Figure 21b).

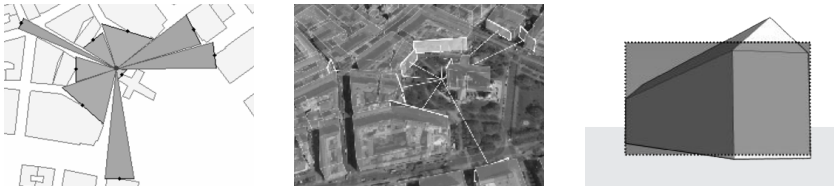
- as a query result, the client gets one or more POIs.

In this regards, the work by Gardiner et. al. [33] is similar to iVu.KOM in that it supports viewer-centric queries that are modeling both the LoS and FoV. However, a major difference is still noteworthy. Thanks to the geometry information included in the iVu model, iVu.KOM allows prompt execution of a query, even if the user's location or heading changes, because communication with the server is not repeatedly needed. However, in the case of [33], when the user moves or changes her heading, the query has to be re-sent to the server for processing. This suggests that the query process in [33] is relevant only for intermittent interaction, i.e. if the user sends infrequent queries. In case of a mobile setting, where the user is continuously moving or changing the device heading or tilt, the query needs to be resent and re-evaluated on the server. On one hand, this can be a downside if real-time experience is required, and on the other hand this can have additional bandwidth costs, which may be undesirable for users with limited mobile connectivity.

4.3.2 *The Work of Simon*

The approach by Simon [57] can be summarized as follows:

- the client on the mobile device reads only the current location ρ and use it as a query parameter.
- the query is sent to the server for processing.
- the server computes both the POIs and the billboards that are visible from the current location. A simple line-of-sight algorithm is used to identify the visible POIs. As shown in



(a) LVis Billboards (top view)

(b) LVis Billboards (3D view)

(c) LVis Billboard perspective distortion

Figure 22: Simplifying geometry to billboards in LVis (source [57])

Figure 22), a billboard of a building represents a rectangle that stands up normal to the ground, is as wide as the visible cross-section of the building and is as high as the building. The billboard faces towards the user and is placed at the intersection point between the center of the visible cross-section and the building footprint.

- as a query result, the client gets the so-called *Local Visibility Model (LVis)* which contains both the POIs and billboards.
- the client reads the heading ϕ and tilt θ and uses them to execute a directional query against the LVis model.

In this respect, the LVis approach is comparable to iVu.KOM in that the query result contains a geometry description besides the POIs. However, it is noteworthy that the computed billboards are location-dependent. Once the user moves, the visibility of the billboards and POIs changes, mandating that a completely new LVis model to be re-collected from the server. In contrast to this, the skyline in iVu.KOM can be fully reconstructed on the client due to the availability of full geometry information (in form of bounding boxes) in the iVu model.

Another major downside of the LVis modeling - as also acknowledged by the author - is that the rectangular billboards can have some mismatch with the actual facades of the buildings due to perspective projection (as shown in Figure 22). In contrast to this, iVu.KOM allows easily extending the definition of the billboard so that errors due to perspective mismatch are reduced or even eliminated. Chapter 5, elaborates on this aspect of iVu.KOM.

Finally, while iVu.KOM provides a user-feedback interface as an explicit means to recognize and react to the achieved query accuracy, the work in [57] focuses only on evaluating the effect of GPS and compass errors on the LVis model.

4.3.3 *Commercial Applications*

The operation of commercial applications like Layar [3] and Wikitude [5] can be summarized as follows:

- the client on the mobile device reads only the current location ρ and uses it as a first query parameter. Another query parameter is the search radius d that the user specifies manually to limit her viewing range (see Figure 23).
- the query is sent to the server for processing.
- the server computes all POIs lying in a circle of radius d around the center ρ.
- as a query result, the client gets a set of POIs.
- the client reads the heading ϕ and tilt θ and use them to execute a directional query against the POIs.

Figure 23: Adjusting search radius in Wikitude [5]

While approaches like [57, 33] represented research efforts on viewer-centric services, Layar and Wikitude have been already adopted on a commercial scale. This can be attributed to the fact that such applications do not integrate world geometry models, which makes these applications easier to develop, maintain and scale. However, the lack of geometry models has the downside that users have to manually adjust the viewing range to limit the search space. Thereby missing one of the viewer-centricity's key elements (as stated in Chapter 1). Unfortunately, it is unclear from the very few literature available about these applications if any means for recognizing or reacting to sensor errors is really implemented. As shown in Figure 23, only a notice about the GPS signal level (and hence GPS accuracy) is displayed. Explicit information about the overall accuracy of the query result is not provided.

Table 2 summarizes the query parameters that are used in the aforementioned approaches.

Table 2: Comparing main approaches for viewer-centric mobile services

Approach	Client-side Query	Server-side Query	Result of Server Query
Gardiner et. al.	–	ρ, θ, ϕ	POIs
Simon	ϕ, θ	ρ	POIs and billboards
Layar/ Wikitude	ϕ, θ	ρ, d	POIs
iVu.KOM	$\bar{\rho}, \phi, \theta$	ρ, d	POIs and bounding boxes

5

EXPLOITING USER'S FEEDBACK IN THE QUERY PROCESS

This chapter presents the function of the Feedback Interface component highlighted in Figure 24. It then presents the design of an augmented reality feedback interface based on the skyline concept.

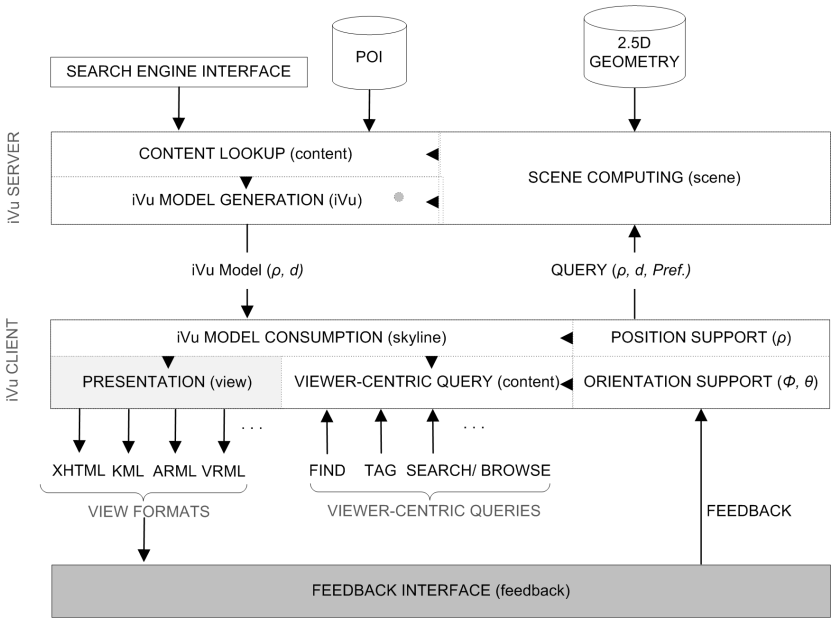


Figure 24: Components covered by this chapter

5.1 DESIGN OF THE FEEDBACK INTERFACE

One main goal of the iVu.KOM framework is to offer the user a feedback interface with support of two primary functions:

- making the user aware of the query accuracy: appropriate display of the uncertainty in the query result can have positive effect on the usability of the application [60]. As iVu.KOM is designed to support different output modalities (e.g. text vs. augmented reality), the feedback interface should make the uncertainty information accessible in a form suitable to the output modality being used. The following sections present a design of a feedback interface for augmented reality based on the skyline concept.

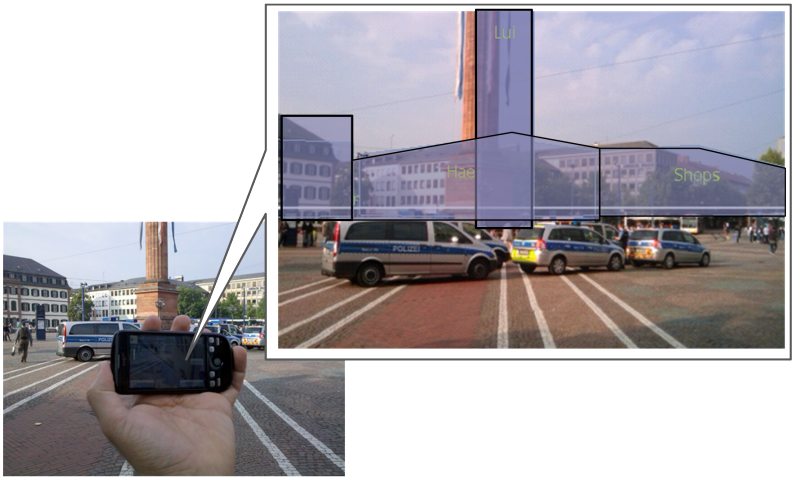


Figure 25: Skyline overlaid on camera image while the user is pointing

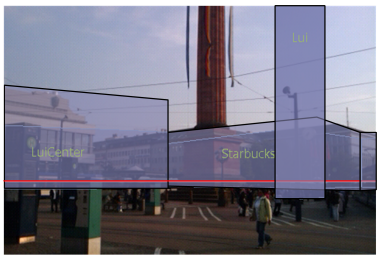
- enabling the user to give feedback about the achieved query accuracy: the feedback can be used to improve the quality of the next queries that are executed under similar conditions.

Exploiting the skyline as an important design resource

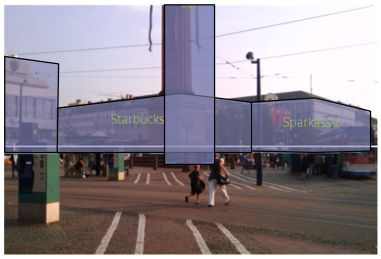
The skyline concept, as explained in Chapter 4, provides a means to perform viewer-centric queries. The skyline can be exploited as well as a design resource for the feedback interface. As shown in Figure 25, the key idea is to superimpose the computed skyline as an image on the top of the camera preview finder while the user is pointing the mobile device towards the target object. This design option is viable nowadays with the availability of camera-enabled smart phones, and the needed API and toolkit support. The interface is based on projecting the camera's viewing port as a window in the skyline, where the width and the height of this window correspond to the horizontal and vertical viewing angles of the camera, respectively. This is analogous to running a point-to-search/browse query as explained in Chapter 4, however, the only difference here is that the compass and tilt readings are used to move the skyline relative to the camera's (fixed) viewing port, whereas in the point-to-search/browse query the FoV is moved with respect to the fixed skyline. Effectively, this gives the user the perception that the camera is moving over the skyline as the user changes the camera direction. The visible part of the skyline is drawn on the camera canvas using the same scale and viewing angle of the camera, which effectively overlays the skyline as another image on top of the camera image.

This design achieves the required interface functions as follows:

- *Displaying query accuracy*: superimposing the skyline on the camera image gives a direct visual clue about achieved query accuracy as depicted in Figure 26. In case of no compass or accelerometer error, each billboard of the skyline should match the underlying object it represents. In case



(a) Skyline with large compass error



(b) Skyline with small compass error

Figure 26: Errors manifest them directly on the overlaid skyline

of a compass error, the skyline will have an offset to the left (for negative heading error) or to the right (for positive heading error). In case of an accelerometer error, the skyline will have an offset upwards (for positive tilt offset) or downwards (for negative tilt offset). GPS errors will generally manifest themselves as a change in the shape, size and placement of the billboards.

- *Providing feedback:* as explained in Chapter 4, neither heading nor tilt values (or errors) affect the shape or placement of the billboard in the skyline. Therefore, it is possible to compensate for compass and accelerometer errors by shifting the skyline horizontally by $\Delta\phi$ degrees or vertically by $\Delta\theta$ degrees, respectively, until it matches with the underlying image. Although it is possible to give this feedback (i.e. to shift the skyline) while the user is pointing the camera towards the target, this can have usability issues as the skyline may show some oscillations, specially in an environment with varying magnetic interference. Therefore, to increase the usability of the interface, the feedback can be given by firstly using the camera to take a standstill snapshot of the surrounding. Then the superimposed skyline can be shifted manually over the image while the user is holding the device in any arbitrary position.

The feedback given by the user is then maintained in a history $\mathcal{H} = \{h_1, \ldots, h_M\}$ where each history sample h_m is specified as:

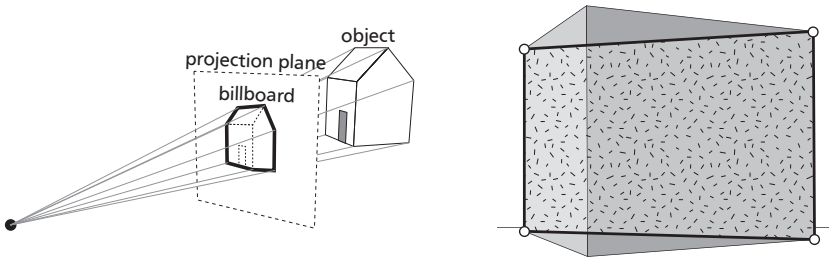
$$h_m = (\{\bar{p}, \phi, \theta\}, \{\Delta\bar{p}, \Delta\phi, \Delta\theta\}) \tag{5.1}$$

where

$\bar{p}$ is the current position,

ϕ is the current (compass) heading,

θ is the current (accelerometer) tilt,



(a) (b)

Figure 27: Difference between (a) a full billboard and (b) a cross-section-based billboard

$\Delta\bar{\rho}$ is the uncertainty in the current position,

$\Delta\phi$ is the heading compensation (introduced by feedback), and

$\Delta\theta$ is the tilt compensation (introduced by feedback).

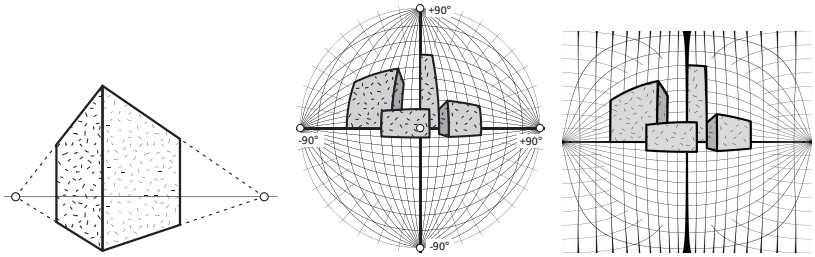
As the behavior of the compass is affected mainly by the magnetic field in the local environment, it is important as well to store the current position with which the compensation factors are associated. Storing the position uncertainty is also needed to specify the range within which the compensation factors can still hold.

5.2 BILLBOARD'S PERSPECTIVE MISMATCH

Billboards are adapted for purposes of an augmented reality-based interface design

It is noticed in Figure 26 that the skyline in the interface has billboards with arbitrary shape and not necessarily rectangular, as already introduced in Chapter 4. The reason for this is that, for the purposes of the feedback interface, rectangular billboards will have poor match with objects in the camera image due to perspective projection. This effect makes it difficult to recognize sensor errors and makes giving the feedback a confusing task.

Figure 27 explains the difference between a billboard resulting as the full projection of a 3D object, and a billboard based on the cross-section of the object, as specified by the skyline concept. Using a cross-section billboard, the billboard may look considerably smaller than the actual visible object, specially when the object is viewed from a corner. Consequently, the visibility of the objects behind that object can be incorrectly computed since the billboard only covers a part of the object. Actually, and as the tests proved, using the rectangular billboards for the design of the interface was found to be pretty much confusing. Therefore, the constructed billboards were extended to include a third anchor point for the case when three faces of the object are visible.



(a) A 2-point perspective (b) A 5-point perspective (c) An equirectangular projection

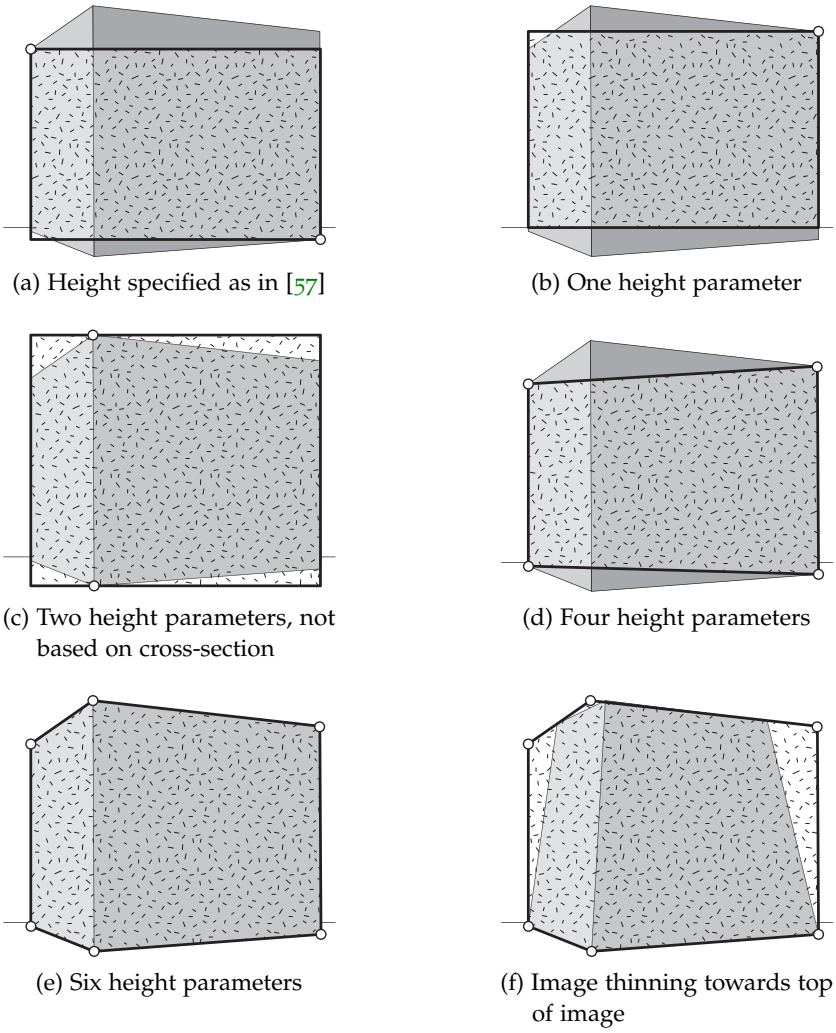
Figure 28: Constructing the billboards using different perspective kinds

To construct a skyline that can match the underlying camera image, a simple 2-point perspective as shown in Figure 28a can be applied. Although this kind of projection yields good results, the accuracy may degrade since the vanishing points of that perspective are only valid for a specific viewing direction. A more correct construction requires a perspective that is independent of the vanishing points. The 5-point perspective as shown in Figure 28b results in a view similar to a photo taken with a fish-eye camera. While the 5-point perspective provides a correct angular description of the perceived scene, an equirectangular projection is more useful to meet the requirements of the interface design. As shown in Figure 28c, an equirectangular projection of a 360° degree panorama results in a uniform rectangular grid of angles, ranging from $-180°$ to $+180°$ horizontally and from $-90°$ to $+90°$ vertically, which still aligns pretty well with the sykline concept introduced in Chapter 4

The equirectangular projection used by the skyline is sufficient for the interface design

While in a camera photo every line seems to maintain its straightness, this is not the case in an equirectangular projection, where only vertical lines remain straight. Horizontal lines are curved since the distance from observer to a straight horizontal line changes. A circle centered at the point of view on the other hand would result in a straight line. For an angular billboard representation of a bounding box this means that the angular height is largest at the point the box has it's minimal distance to the observer. This effect is not visible in most cases, specially if the object is completely visible in the field of view. This effect can be noticed if the observer stands rather close on the side of a long building. Here, a visualized billboard will seem too small to fit the building, similar to the effect encountered in the cross-section-based billboards.

The design goal for iVu's billboards is to be as accurate as possible while being as simple as possible. A left and right heading angle that limits the horizontal extensions of a billboard was obvious. The interesting part is the vertical extension. The billboards in [57] uses in effect a single upper and lower angle (defined by a



(a) Height specified as in [57]

(b) One height parameter

(c) Two height parameters, not based on cross-section

(d) Four height parameters

(e) Six height parameters

(f) Image thinning towards top of image

Figure 29: Billboard accuracy vs. number of height parameters used

center angle in conjunction with a vertical height) which results in a rectangular billboard as shown in Figure 29a. However, in the case that the observer is simply placed on the ground, this can be simplified to just one angle above the horizon as shown in Figure 29b. A drawback here is that the ground elevation can not be considered. If defined only by an upper and lower angles, there are multiple possibilities where to place these two. For example, for the upper angle any point between the lowest upper point (at the most distant corner) and the highest point (at the closest point) can be used as shown in Figure 29c. For buildings viewed from a corner none of these possibilities will be very accurate, as the billboard will be either too small in some areas or too large others.

Achieving billboard accuracy along the skyline tilt axis

To achieve a more accurate result, more than one height needs to be defined. An accurate model for cross-section based billboards (excluding the curved lines) is to use four parameters for

the vertical extension: for both the left and right corner an upper and lower angle 29d. For the purposes of the interface design, iVu.KOM goes one step further by extending the billboard by a third pair of parameters for the height of the middle corner as depicted in Figure 29e.

While the above model is correct from an angular point of view, objects in the the camera image can still deviate from the computed billboards. In general, cameras introduce additional perspective distortions that do not exist in the above angular model. A common example is the perceived thinning towards the top of buildings when the camera is tilted upwards as depicted in Figure 29f. In the current design, these distortions are ignored as they showed very minor effect on the performed viewer-centric queries.

The feedback interface designed with these considerations proved as well very useful to evaluate the performance of the iVu.KOM framework as will be explained in Chapter 8